Análisis y diseño de algoritmos

JUAN BERNARDO VAZQUEZ GOMEZ

Red Tercer Milenio

Análisis y diseño de algoritmos

JUAN BERNARDO VAZQUEZ GOMEZ

Red Tercer Milenio



AVISO LEGAL

Viveros de Asís 96, Col. Viveros de la Loma, Tlalnepantla, C.P. 54080, Estado de México.

Datos para catalogación bibliográfica

Juan Bernardo Vázquez Gómez

Análisis y diseño de algoritmos

ISBN 978-607-733-053-0

Primera edición: 2012

DIRECTORIO

PROPÓSITO GENERAL

El alumno adquirirá y aplicará los conocimientos que le permitan plantear una metodología para la solución de problemas, utilizando la computadora a través del diseño de algoritmos.

INTRODUCCIÓN

En la actualidad el ser humano se enfrenta a distintas situaciones: de aprendizaje, de retroalimentación y en muchas ocasiones dificultades que con la experiencia o por la elección de la alternativa apropiada, va dando solución.

Es por ello que a menudo se emplea cierta metodología para la solución de los problemas en lugar de actuar de forma imprevista, siendo una característica relevante el análisis de los mismos. La humanidad de forma natural emplea en la vida cotidiana ciertas conductas que son rutinarias, siguen un orden, una secuencia y pretenden alcanzar un objetivo.

Este conjunto de acciones rutinarias que se llevan a cabo y forman parte ya de la vida cotidiana del ser humano, se conocen como algoritmos, los cuales son aplicables en los ámbitos que así se necesiten.

El ámbito de mayor de aplicación y de primordial importancia es en la solución de problemas mediante computadora. Donde el elemento base para lograr dicha solución es el algoritmo propio.

Desarrollar un algoritmo involucra tener un conocimiento base sobre las características y elementos que debe contener, con el fin de cumplir sus cualidades: finito, definido y preciso.

Es importante señalar que en el ámbito de la programación antes de resolver el problema mediante la computadora se recomienda realizar primero el algoritmo, ya que es aquí donde se encuentra la solución universal de la problemática en cuestión.

Y para codificar dicho algoritmo sólo basta en adaptar cada uno de sus elementos, al lenguaje de programación en el que se desee implementar.

Para resolver un problema se pueden desarrollar diversos algoritmos, existen en ocasiones múltiples soluciones, pero dentro de ellas existen las que son más eficientes y es aquí donde la habilidad del desarrollador juega un papel importante. Por lo que la práctica continua contribuye a la mejora del desarrollo.

PROGRAMA DE ESTUDIOS

OBJETIVO GENERAL.

Aplicar los conocimientos que le permitan plantear una metodología para la solución de problemas, utilizando la computadora a través del diseño de algoritmos.

Temario.

UNIDAD 1. CONCEPTOS BÁSICOS.

1.1 DEFINICIÓN DE ALGORITMOS, LENGUAJE Y APLICACIONES
1.2 DEFINICIÓN Y SOLUCIÓN DE PROBLEMAS
1.3 ALGORITMOS COTIDIANOS
1.4 LENGUAJE DE PROGRAMACIÓN
1.4.1 Clasificación de los lenguajes de programación
1.4.2 Componentes de los lenguajes de programación
1.5 INTRODUCCIÓN AL LENGUAJE C++

UNIDAD 2. EL ALGORITMO COMO ESTRATEGIA Y/O HERRAMIENTA PARA LA SOLUCIÓN DE PROBLEMAS.

2.1 ANÁLISIS PARA LA SOLUCIÓN DE UN PROBLEMA
2.2 TIPOS DE DATOS
2.2.1 Tipos de datos en C++
2.3 DISEÑO DE UN ALGORITMO.
2.4 TIPOS DE PROGRAMACIÓN

UNIDAD 3. ESTRUCTURAS BÁSICAS DE UN ALGORITMO.

3.1 ESTRUCTURAS SECUENCIALES

3.2 ESTRUCTURAS CONDICIONALES

3.3 CONTADORES

3.4 ACUMULADORES

3.5 CICLOS

3.5.1 While

3.5.2 Do…while

3.5.3 For

3.6 ARREGLOS

UNIDAD 4. MODULARIDAD.

4.1 CONCEPTO

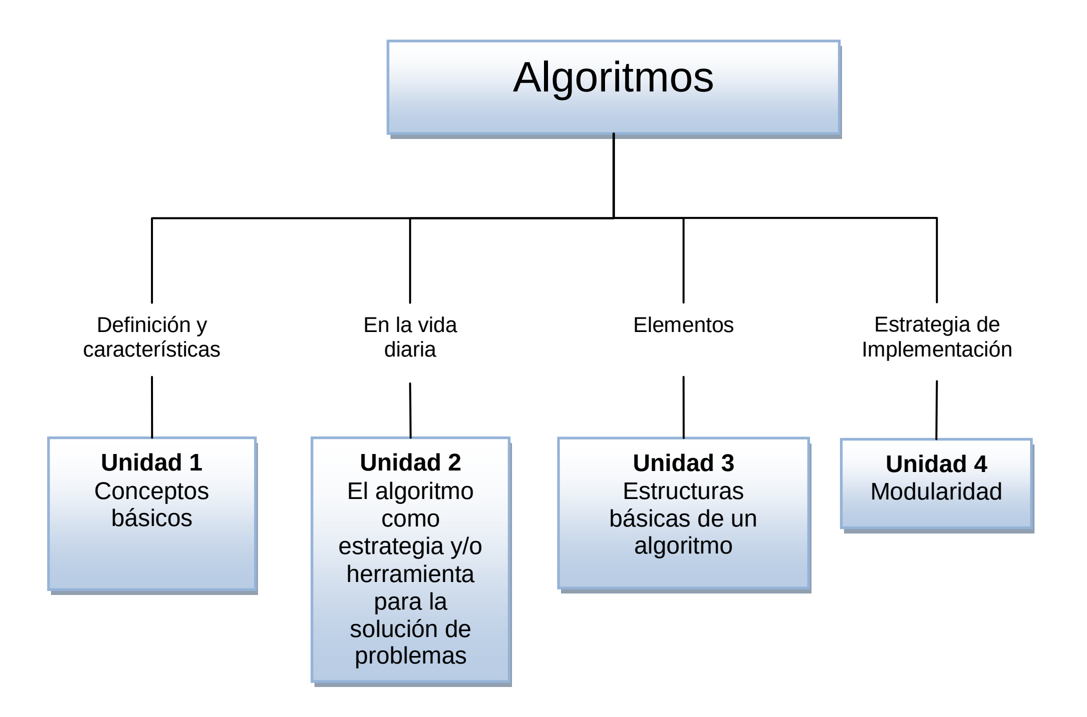
4.2 CARACTERÍSTICAS

4.3 COMPONENTES

4.4 COMUNICACIÓN

4.5 TIPOS

MAPA CONCEPTUAL



INDICE

UNIDAD 1

CONCEPTOS BÁSICOS



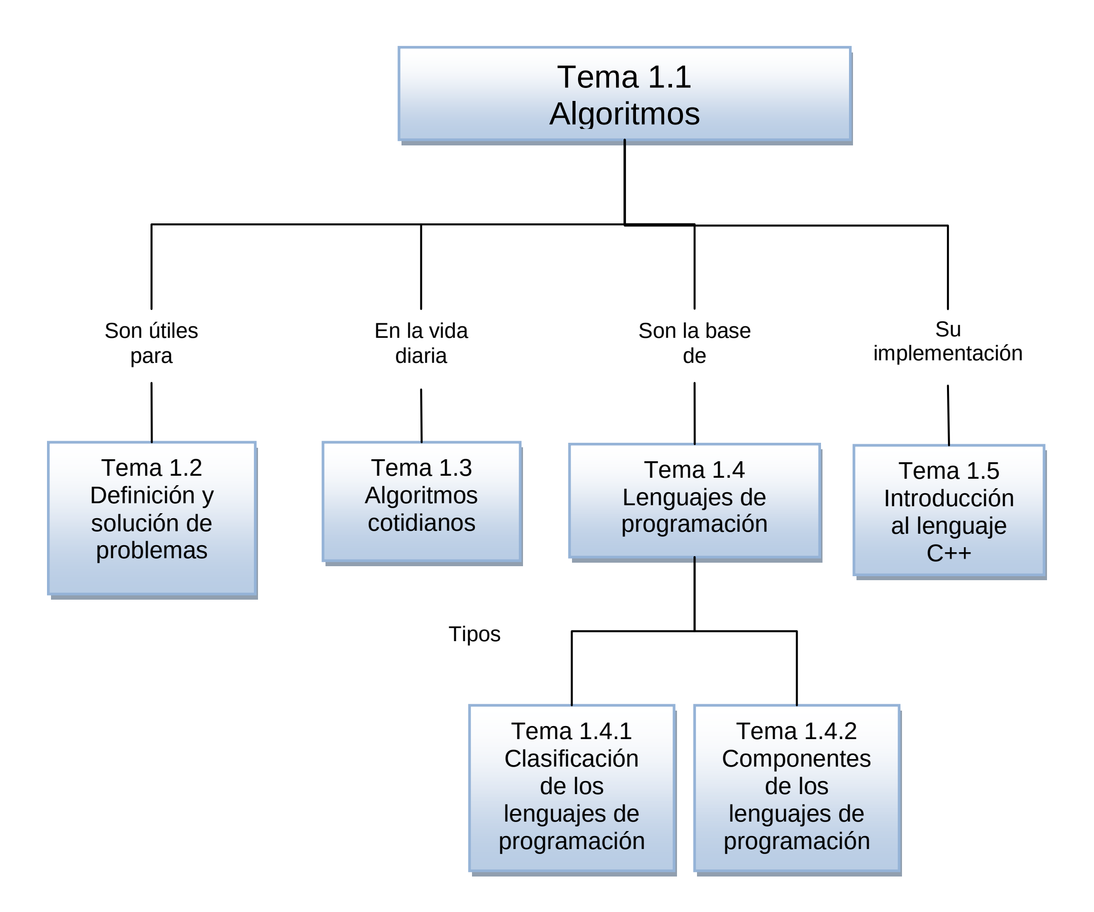
www.fondosescritorio.net/.../Via-Lactea-2.jpg

OBJETIVO

El alumno definirá el concepto de algoritmo. Analizará el lenguaje como medio de comunicación, para establecer los parámetros necesarios en el desarrollo de un algoritmo.

TEMARIO

MAPA CONCEPTUAL



INTRODUCCIÓN

Al resolver problemas de distinta índole, en muchas ocasiones se sigue una metodología para conseguir tal propósito. Dicha metodología se encuentra caracterizada por una serie de acciones o situaciones llevadas a cabo.

Las cuales tienen como propósito, cada una de ellas lograr un objetivo en específico y en conjunto alcanzar un objetivo general. En consecuencia, estas acciones se pueden concebir como algoritmos, por lo que resulta importante conocer la definición, características e importancia que tienen los algoritmos en los distintos ámbitos de la vida cotidiana.

De la misma manera los algoritmos son una herramienta vital para la solución de problemas mediante computadora; es por ello que esta unidad pretende dar a conocer los conceptos básicos de los algoritmos.

1.1. DEFINICIÓN DE ALGORITMOS, LENGUAJE Y APLICACIONES

Objetivo

El participante definirá el concepto de algoritmos y lenguaje e identificar sus aplicaciones.

En su libro *Fundamentos de programación*, Luis Joyanes Aguilar, define al algoritmo como un método para resolver un problema. Aunque la popularización del término ha llegado con el advenimiento de la era informática, algoritmo proviene de *Mohammed al-KhoWârizmi*, matemático persa que vivió durante el siglo IX y alcanzó gran reputación por el enunciado de las reglas paso a paso para sumar, restar, multiplicar y dividir números decimales; la traducción al latín del apellido en la palabra *algorismus* derivó posteriormente en algoritmo. Señala, Joyanes Aguilar, que Euclides, matemático griego (del siglo IV a.C.) quien inventó un método para encontrar el máximo común divisor de dos números, se considera con Al-KhoWârizmi el otro gran padre de la algoritmia (ciencia que trata de los algoritmos).[1]

Joyanes Aguilar, hace un señalamiento histórico con respecto a Niklaus Wirth, inventor de *Pascal, Modula-2 y Oberon*, profesor quien tituló uno de sus más famosos libros, *Algoritmos+Estructuras de datos=Programas*, señalandonos que sólo se puede llegar a realizar un buen programa con el diseño de un algoritmo y una correcta estructura de datos.

[1] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. pp. 52-53.

La resolución de un problema exige el diseño de un algoritmo que resuelva el mismo. La propuesta para la resolución de un problema es la siguiente:

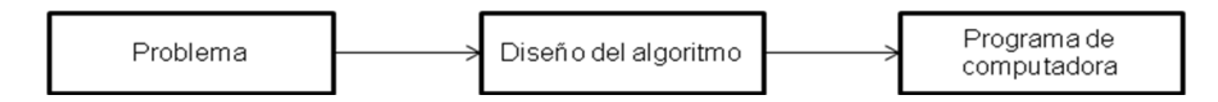


Figura 1. Resolución de un problema. [1]

Los pasos para la resolución de un problema son:

1.- *Diseño del algoritmo*, describe la secuencia ordenada de pasos, sin ambigüedades, que conducen a la solución de un problema dado. *(Análisis del problema y desarrollo del algoritmo).*

2.- Expresar el algoritmo como un *programa* en un lenguaje de programación adecuado. *(Fase de codificación).*

3.- *Ejecución y validación* del programa por computadora.

Para llegar a la resolución de un problema es necesario el diseño previo de un algoritmo, de modo que sin algoritmo no puede existir un programa.

Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que los ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y ejecutarse en una computadora distinta; sin embargo, el algoritmo será siempre el mismo. Así, por ejemplo, en una analogía con la vida diaria, la receta de un platillo de cocina se puede expresar en español, inglés o francés, pero cualquiera que sea el lenguaje, los pasos para la elaboración, del mismo se realizarán sin importar el idioma del cocinero.

En la ciencia de la computación y en la programación, los algoritmos son más importantes que los lenguajes de programación o las computadoras. Un

lenguaje de programación es tan sólo un medio para expresar un algoritmo y una computadora es sólo un procesador para ejecutarlo. Tanto el lenguaje de programación como la computadora son los medios para obtener un fin: conseguir que el algoritmo se ejecute y se efectúe el proceso correspondiente.

Dada la importancia del algoritmo en la ciencia de la computación, un aspecto importante será el *diseño de algoritmos.*

Joyanes Aguilar, enfatiza que el diseño de la mayoría de los algoritmos requiere creatividad y conocimientos profundos de la técnica de programación. En esencia, *la solución de un problema se puede expresar mediante un algoritmo.* [2]

Características de los algoritmos

Las características fundamentales que debe cumplir todo algoritmo son:

- Un algoritmo debe ser *preciso* e indicar el orden de realización de cada paso.
- Un algoritmo debe estar *definido*. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser *finito*. Si se sigue un algoritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos.

La definición de un algoritmo debe describir tres partes: *Entrada, Proceso y Salida.* Por ejemplo, en el cálculo de la edad de una persona, conociendo su año de nacimiento, la definición del algoritmo, quedaría de la siguiente manera:

Entrada: la edad de la persona, información del año de nacimiento y el actual.

Proceso: realizar la diferencia del año actual menos el año de nacimiento.

Salida: visualización del resultado generado. Es decir, el resultado es la edad.

Escritura de Algoritmos

[2] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 53.

Se emplea un lenguaje natural describiendo paso a paso el algoritmo en cuestión.

En la realización del algoritmo aplicar sus tres características: preciso, definido y finito.

Por ejemplo algoritmo para conocer si el promedio de un alumno es aprobatorio o no teniendo como referencia que alumno cursa 5 materias y además que el promedio mínimo aprobatorio es 7.0.

1.- **inicio**
2.- solicitar las cinco calificaciones del alumno
3.-sumar las cinco calificaciones del alumno
4.- El resultado del paso 3 dividirlo entre cinco
5.- **si** el resultado del paso 4 es mayor o igual a 7.0 **entonces**
    5.1Visualizar Alumno aprobado
    si_no
    5.2 Visualizar Alumno reprobado
    fin_si
6.- **fin**

En el algoritmo anterior se dio solución al planteamiento básico del cálculo del promedio de un alumno. Se observa que los pasos del algoritmo tienen un número que conforme se va describiendo la secuencia, ese número se va incrementando. Es importante destacar que todo algoritmo es finito, es decir, así como tiene un inicio debe tener un fin, lo que se observa en los pasos 1 y 6.

El lenguaje que se emplea es de lo más natural. En el ejemplo se ilustra la precisión de cada una de las actividades, no se prestan a confusión. Así mismo podrá notarse que en el ejemplo, por su naturaleza existió la necesidad de tomar decisiones, es por ello el empleo de las palabras reservadas (**si-entonces-sino** *if-then-else)* las cuales se emplean para la selección o toma de decisiones.

En el paso 5.1 y 5.2 se implementa lo que se conoce como (sangrado o justificación) en la escritura de algoritmos, que no es más que una tabulación, estrategia recomendada tanto en la escritura de éstos, como en la captura de programas de computadora, ya que faciliten la lectura y permite un análisis más fluido de lo escrito.

Representación gráfica de los algoritmos

Para la representación gráfica del algoritmo debe emplearse un método que sea independiente del lenguaje de programación elegido.

Joyanes Aguilar señala que para conseguir este objetivo se precisa que el algoritmo sea representado gráfica o numéricamente, de modo que las sucesivas acciones no dependan de la sintaxis de ningún lenguaje de programación, sino que la descripción se emplee para su transportación en un programa.

Los métodos usuales para representar un algoritmo son:

1.- Diagrama de flujo.
2.- Diagrama N-S (Nassi-Schneiderman).
3.- Lenguaje de especificación de algoritmos: pseudocódigo.
4.- Lenguaje español, inglés.
5.- Fórmulas.

El método 4 y 5 no son fáciles de programar. Un algoritmo no puede ser representado por una simple fórmula.

Diagrama de flujo

También conocido como *flowchart* es una técnica de programación de representación de algoritmos antigüa y muy utilizada. Un diagrama de flujo, Jayanes Aguilar, lo define como: “un diagrama que utiliza los símbolos (cajas)

estándar mostrados en la tabla 1 y que tiene los pasos de un algoritmo escritos en esas cajas unidas por flechas, denominadas *líneas de flujo*, que indican la secuencia en que se debe ejecutar". [3]

Tabla 1. Símbolos de diagramas de flujo.

Símbolo	Función
(elipse)	Terminal (representa el comienzo, "inicio" y el final, "fin" de un programa. Puede representar también una parada o interrupción programada que sea necesario realizar en un programa.)
(paralelogramo)	Entrada/Salida (cualquier tipo de introducción de datos en la memoria desde los periféricos, "entrada", o registro de la información procesada en un periférico, "salida".)
(rectángulo)	Proceso (cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones matemáticas, de transferencia, etc.).
NO SÍ	Decisión (indica operaciones lógicas o de comparación entre datos, normalmente dos, y en función del resultado de la misma determina cuál de los distintos caminos alternativos del programa se debe seguir; normalmente tiene dos salidas, respuestas SÍ o NO, pero puede tener tres o más, según los casos).

[3] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 59

	Decisión múltiple (en función del resultado de la comparación se seguirá uno de los diferentes caminos de acuerdo con dicho resultado.).
	Conector (sirve para enlazar dos partes cualesquiera de un ordinograma a través de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en la misma página del diagrama.
	Indicador de dirección o línea de flujo (indica el sentido de ejecución de las operaciones.).
	Línea conectora (sirve de unión entre dos símbolos).
	Conector (conexión entre dos puntos del ordinograma situado en diferentes páginas.)
	Llama subrutina o a un proceso predeterminado (una subrutina es un módulo independiente del programa principal, que recibe una entrada procedente de dicho programa, realiza una tarea determinada y regresa, al terminar, al programa principal.)
	Pantalla (se utiliza en ocasiones en lugar del símbolo de entrada/salida).
	Impresora (se utiliza en ocasiones en lugar del símbolo de entrada/salida).

	Teclado (se utiliza en ocasiones en lugar del símbolo de entrada/salida).
	Comentarios (se utiliza para añadir comentarios clasificadores a otros símbolos del diagrama de flujo. Se pueden dibujar a cualquier lado del símbolo.).

La figura 2 es un diagrama de flujo básico. Los símbolos estándar normalizados por ANSI (abreviatura de *American National Standars Institute)* son muy variados.

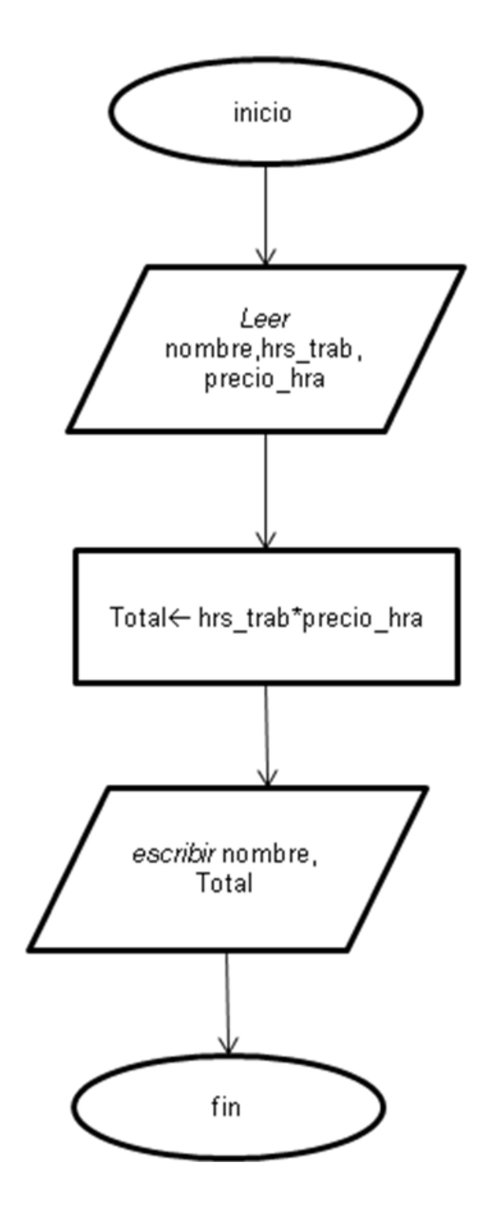


Figura 2. Diagrama de flujo

Cada símbolo visto previamente indica el *tipo de operación a ejecutar* y el diagrama de flujo ilustra gráficamente la secuencia *en la que se ejecutan las operaciones.*

ACTIVIDADES DE APRENDIZAJE

1.- Definir el proyecto a realizar durante el semestre. El catedrático orienta al grupo sobre los proyectos a realizar. El producto final a entregar del proyecto será desarrollar el diagrama de flujo de un sistema, por ejemplo un sistema de ventas, con sus opciones básicas (agregar productos, nota de venta, factura.)

2.- Realizar un cuadro sinóptico sobre el concepto de algoritmos, considerar cinco autores. Identificar las similitudes y diferencias entre las definiciones. Mínimo 1 cuartilla. La entrega de la actividad será en forma impresa. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

3.- Realizar un cuadro sinóptico en donde se defina el concepto de lenguaje y sus aplicaciones. Mínimo una cuartilla. La entrega de la actividad será impresa. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

1.2. DEFINICIÓN Y SOLUCIÓN DE PROBLEMAS.

Objetivo

Se explicará y ejemplificará la metodología para la definición y solución de problemas.

Fase en la resolución de problemas[4]

Joyanes Aguilar, señala que el proceso de resolución de problemas con una computadora conduce a la escritura de un programa y a su ejecución en la misma. Reconoce que el proceso de diseño de un programa es un “proceso creativo”, en el cual se puede considerar una serie de fases o pasos comunes, que generalmente deben seguir los programadores.

Las fases de resolución de un problema con computadora son:

- Análisis del problema.
- Diseño del algoritmo.
- Codificación.
- Compilación y ejecución.
- Verificación.
- Depuración.
- Mantenimiento.
- Documentación.

Constituyen el ciclo de vida del software y las fases o etapas usuales son:

[4] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p.40.

1. *Análisis.* El problema se analiza teniendo presente la especificación de los requisitos dados por el cliente de la empresa o por la persona que se encarga del programa.

2. *Diseño.* Una vez analizado el problema, se diseña una solución que conduzca a un *algoritmo* que resuelva el problema.

3. *Codificación (implementación).* La solución se escribe en la sintaxis del lenguaje de alto nivel (por ejemplo, C++) y se obtiene un programa.

4. *Compilación, ejecución y verificación.* El programa se ejecuta, se comprueba rigurosamente y se eliminan todos los errores que puedan aparecer.

5. *Depuración y mantenimiento.* El programa se actualiza y modifica cada vez que sea necesario, de modo que se cumplan todas las necesidades de cambio de sus usuarios.

6. *Documentación.* Escritura de las diferentes fases del ciclo de vida del software, específicamente, el análisis, diseño y codificación, unidos a manuales de usuario y de referencia, así como normas para el mantenimiento.

ACTIVIDADES DE APRENDIZAJE

1.- Realizar un cuadro sinóptico sobre la definición y solución de problemas. Mínimo una cuartilla. La entrega de la actividad será impresa. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

2.- Resolver el siguiente ejercicio: se desea calcular el promedio de un estudiante quien tiene 5 materias. Emplear la metodología para la definición y solución de problemas. Señalar los datos de entrada, el proceso y los datos de salida.

3.- Resolver el siguiente ejercicio: se desea calcular el promedio general de un grupo que tiene un total de diez estudiantes los cuales tiene 4 materias. Emplear la metodología para la definición y solución de problemas. Señalar los datos de entrada, el proceso y los datos de salida. La entrega será impresa. Considerar la limpieza y ortografía.

4.- Resolver el siguiente ejercicio: se desea calcular el promedio de dos estudiantes cada uno tiene 10 materias. Emplear la metodología para la definición y solución de problemas. Señalar los datos de entrada, el proceso y los datos de salida.

5.- Resolver el siguiente ejercicio: se desea calcular el promedio general de un grupo que tiene un total de quince estudiantes los cuales tienen 7 materias. Emplear la metodología para la definición y solución de problemas. Señalar los datos de entrada, el proceso y los datos de salida. La entrega será impresa. Considerar la limpieza y ortografía.

1.3 ALGORITMOS COTIDIANOS

Objetivo

El alumno podrá identificar y explicar las características de los algoritmos más comunes.

Algoritmos cotidianos

Se refiere a los algoritmos que ayudan a resolver problemas diarios, y que las personas llevan a cabo sin darse cuenta de que están siguiendo una metodología para resolverlos.

Algunos ejemplos son:

- Diseñar un algoritmo para cambiar una llanta a un auto:

1. Inicio.
2. Conseguir un gato hidráulico.
3. Levantar el auto con el gato.
4. Aflojar los tornillos de las llantas.
5. Retirar los tornillos de las llantas.
6. Quitar la llanta.
7. Colocar la llanta de repuesto.
8. Colocar los tornillos.
9. Apretar los tornillos.
10. Bajar el gato hidráulico.
11. Retirar el gato hidráulico.
12. Fin

- Diseñar un algoritmo que compare el mayor de dos números

1. Inicio
2. Obtener el primer número (entrada), denominado NUMERO1.
3. Obtener el segundo número (entrada), denominado NUMERO2.
4. Si NUMERO1 es igual a NUMERO 2 entonces

   3.1 Visualizar “son iguales”

4. Si NUMERO1 es mayor a NUMERO2 entonces

   4.1 NUMERO1 es mayor

   4.2 SINO

   4.3 NUMERO2 es mayor

5. Fin

- Diseñar un algoritmo que permita obtener un refresco de una máquina automática expendedora de bebidas embotelladas.

1. Inicio
2. Verificar el panel de bebidas, ubicando la bebida deseada.
3. Identificar el costo de la bebida.
4. Introducir en la ranura correspondiente la cantidad monetaria que así corresponda a la bebida deseada, de preferencia introducir la cantidad exacta.
5. Pulsar el botón que corresponda a bebida deseada.
6. Si existe producto entonces
    a. En la bandeja de salida saldrá la bebida seleccionada.
    b. Sino
    c. En el panel se visualizará producto agotado.
7. Si cantidad introducida es igual al precio de producto entonces
    a. No devolverá cambio.
8. Si cantidad introducida es mayor al precio de producto entonces
    a. Devolverá el efectivo de diferencia en la bandeja pertinente.
9. Si cantidad introducida es menor al precio de producto entonces
    a. En el panel visualizará efectivo insuficiente.
10. Fin

ACTIVIDADES DE APRENDIZAJE

1.- Realizar una síntesis de los algoritmos más comunes (cambiar una llanta, calcular el promedio, calcular la edad de una persona, preparar algún platillo de cocina). Entrega impresa. Especificar bibliografía consultada. Considerar la limpieza y ortografía. Mínimo tres ejemplos.

1.4. LENGUAJES DE PROGRAMACIÓN.

Objetivo

El alumno será capaz de definir el concepto de lenguaje de programación.

Lenguajes de Programación

Joyanes Aguilar señala que los *lenguajes de programación* sirven para escribir programas que permitan la comunicación usuario/máquina. Unos programas especiales llamados *traductores* (*compiladores o intérpretes*) convierten las instrucciones escritas en lenguajes de programación en instrucciones escritas en lenguajes máquina (0 y 1, bits) que ésta pueda entender.

Agrega que los *programas de utilidad* facilitan el uso de la computadora. Un buen ejemplo es un *editor de textos* que permite la escritura y edición de documentos.

Los programas que realizan tareas concretas; nóminas, contabilidad, análisis estadístico, etc., se denominan *programas de aplicación*. [5]

Peter Norton, en su libro titulado *"Introducción a la Computación"*, menciona que programar es una manera de enviar instrucciones a la computadora. Para estar seguros de que la computadora (y otros programadores) pueden entender esas instrucciones, los programadores usan lenguajes definidos para comunicarse.

Estos lenguajes tienen reglas del tipo que la gente usa para comunicarse entre sí. Por ejemplo, cita, la información que debe ser proporcionada en un cierto orden y estructura, se usan los símbolos y con frecuencia se requiere información.

---

[5] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 21.

El único lenguaje que una computadora comprende es su **lenguaje máquina.** Sin embargo, la gente tiene dificultad para entender el código máquina. Como resultado, los investigadores desarrollaron primero un lenguaje ensamblador luego lenguajes de nivel superior. Esta evolución representa una transición de hileras de números (código máquina) a secuencias de comandos que se pueden leer como cualquier otro lenguaje. Los lenguajes de nivel superior se enfocan en lo que el programador quiere que haga la computadora, no en cómo la computadora ejecutará esos comandos.[6]

1.4.1. Clasificación de los Lenguajes de programación

Norton, señala la siguiente clasificación:

Lenguajes máquina: son los lenguajes básicos. Consisten en hileras de números y son definidos por el diseño del hardware. En otras palabras, el lenguaje máquina para una Macintosh no es el mismo que el de una PC. Una computadora comprende sólo su lenguaje máquina original, los comandos de su equipo de instrucción. Estos comandos le dan instrucciones a la computadora para realizar operaciones elementales: cargar, almacenar, añadir y sustraer. Esencialmente, el código máquina consiste por completo de los 0 y 1 del sistema numérico binario.

Lenguajes ensambladores, fueron desarrollados usando nemotécnicos similares a las palabras del idioma inglés. Los programadores trabajan en editores de texto, que son simples procesadores de palabras, para crear *archivos fuente.* Los archivos fuente contienen instrucciones para que la computadora las ejecute, pero tales archivos deben primero introducirse al lenguaje máquina. Los investigadores crearon programas traductores llamados *ensambladores*

[6] NORTON, Peter. Introducción a la Computación. Mc Graw Hill. México. 3ra edición. 2000. p. 427.

para realizar la conversión. Los lenguajes ensambladores aún son altamente detallados y secretos, pero leer un código ensamblador es mucho más rápido que batallar con el lenguaje máquina. Los programadores rara vez escriben programas de tamaño significativo en un lenguaje ensamblador. (Una excepción a esta regla se encuentra en los juegos de acción en donde la velocidad del programa es decisiva.) En su lugar, se usan lenguaje ensambladores para afinar partes importantes de programas escritos en un lenguaje de nivel superior.

Los *lenguajes de alto nivel* fueron desarrollados para hacer más fácil la programación. Estos lenguajes son llamados de alto nivel porque su sintaxis es más cercana al lenguaje humano que el código del lenguaje máquina o ensamblador. Usan palabras familiares en lugar de comunicar en el detallado embrollo de los dígitos que comprenden las instrucciones de la máquina. Para expresar las operaciones de la computadora estos lenguajes usan operadores, como los símbolos de más o menos, que son los componentes familiares de las matemáticas. Como resultado, leer, escribir y comprender programas de cómputo es más fácil con un programa de alto nivel, a pesar de que las instrucciones todavía deba ser introducidas al lenguaje máquina antes de que la computadora pueda comprenderlas y llevarlas a cabo.

Los comandos escritos en cualquier lenguaje ensamblador o de alto nivel deben ser traducidos de nuevo a código máquina antes de que la computadora pueda ejecutar los comandos. Estos programas traductores se denominan *compiladores.* Entonces, normalmente un programa debe ser compilado o traducido a código máquina antes de que se ejecute. Los archivos de programas compilados se vuelven ejecutables.

A continuación se recaba algunos de los lenguajes de programación de alto nivel más importantes.

Lenguajes de alto nivel

Los lenguajes de programación son tratados a veces en términos de generaciones. Se considera que cada generación sucesiva contiene lenguajes que son más fáciles de usar y más poderosos que los de la generación previa. Los lenguajes máquina son considerados de la primera generación, y los ensambladores de segunda generación. Los lenguajes de alto nivel comenzaron en la tercera generación.

Lenguajes de tercera generación

Los lenguajes de tercera generación, señala Norton, tienen la capacidad de soportar programación estructurada, lo cual significa que proporcionan estructuras explícitas para ramas y ciclos. Además, debido a que son los primeros lenguajes que usan fraseo similar al inglés, compartir el desarrollo entre los programadores también es más fácil. Los integrantes del equipo pueden leer el código de cada uno de los demás y comprender la lógica y el flujo de control del programa.

Estos programas también son *portátiles.* En oposición a los lenguajes ensambladores, los programas en estos lenguajes pueden ser compilados para ejecutarse en numerosos CPU.

Los lenguajes de tercera generación incluyen:

- FORTRAN
- COBOL
- BASIC
- PASCAL
- C
- C++
- JAVA

Lenguajes de cuarta generación

Los lenguajes de cuarta generación (4GL) son principalmente lenguajes de programación para propósitos especiales, que son más fáciles de usar que los de tercera generación. Con los 4GL los programadores pueden crear aplicaciones rápidamente. Como parte del proceso de desarrollo, los programadores pueden usar los 4GL para desarrollar prototipos de una aplicación rápidamente. Los prototipos dan a los equipos y clientes una idea de cómo se vería y funcionaría la aplicación antes de que el código este terminado[7].

Como resultado, cada uno de los involucrados en el desarrollo de la aplicación puede proporcionar retroalimentación sobre aspectos estructurales y de diseño al principio del proceso.

Una sola declaración en un 4GL logra mucho más de lo que era posible con una declaración similar en un lenguaje de generación anterior. A cambio de esta capacidad de trabajar más rápido, los programadores han demostrado disposición para sacrificar parte de la flexibilidad disponible con los lenguajes anteriores.

Muchos 4GL tienen capacidad para bases de datos, lo que significa que se puede crear con ellos programas que actúen como enlaces con bases de datos.

Dentro de los lenguajes de cuarta generación se incluyen:

- Visual Basic
- Lenguajes de macros específicos para una aplicación
- Ambientes de autoría

Lenguajes de quinta generación[8]

Norton, señala que la quinta generación de los lenguajes de cómputo incluye inteligencia artificial y sistemas expertos. Estos sistemas tienen por objeto pensar y anticipar las necesidades de los usuarios, en lugar de sólo ejecutar un

---

[7] Norton, Peter. Introducción a la Computación. Mc Graw Hill. México. 3ra edición. 2000. p. 429.
[8] Norton, Peter. Introducción a la Computación. Mc Graw Hill. México. 3ra edición. 2000. p. 431.

conjunto de órdenes. A pesar de que los sistemas de inteligencia artificial se están volviendo más difíciles de desarrollar de lo que se esperaba originalmente, lo expertos afirmaban que los sistemas, al igual que las redes nerviosas, pronto serán capaces de tomar hechos y luego usar un conjunto de datos para formular una respuesta apropiada, exactamente como lo realiza el ser humano.

1.4.2 Componentes de los lenguajes de programación

Intérpretes

Joyanes Aguilar, define a un *Intérprete,* como un traductor que toma un programa fuente, lo traduce, y a continuación lo ejecuta. Los programas intérpretes clásicos, como BASC, prácticamente ya no se utilizan, aunque se puede encontrar alguna computadora que opere con la versión QBasic bajo el Sistema Operativo DOS que corre en las computaras personales. Sin embargo, está muy extendida la versión interpretada del Lenguaje Smalltalk, un lenguaje orientado a objetos puro. Los intérpretes han vuelto a renacer con la aparición de Java, ya que para entender el código en bytes *(bytecode)* al que traduce un compilador se necesita un intérprete.

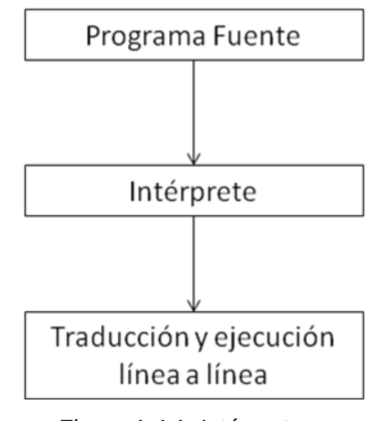


Figura 1.4.1. Intérprete

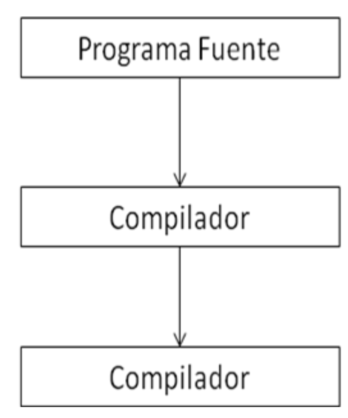


Figura1.4.2. La compilación de programas.

Compiladores

Después que se ha diseñado el algoritmo y escrito el programa en un papel, se debe comenzar el proceso de introducir el programa en un archivo (fichero) en el disco duro de la computadora. La introducción y modificación del programa en un archivo se hace utilizando un *editor de texto* o simplemente un *editor*, un programa que viene en la computadora. El aprendizaje de cómo utilizar un editor hace la tarea de introducir un programa una tarea muy fácil.

El programa ya sea escrito en C o en Java, o en cualquier otro lenguaje, pero ni C ni Java son lenguajes máquina, por el contrario son lenguajes de alto nivel diseñados para hacer más fácil la programación que utilizando el lenguaje máquina. La computadora no entiende los lenguajes de alto nivel. En consecuencia, un programa escrito en un lenguaje de alto nivel debe ser traducido a un lenguaje que la máquina pueda comprender. Los lenguajes que la computadora puede comprender se llaman *lenguajes de bajo nivel.* La traducción de un programa escrito en un lenguaje de alto nivel, como C++ o Java, a un lenguaje que pueda entender la computadora se hace mediante otro programa conocido como *compilador.*

Los lenguajes de bajo nivel que la computadora puede entender directamente se conocen como *lenguajes ensamblador o lenguaje máquina.* En realidad, aunque son muy similares y en ocasiones se les considera sinónimos, tienen algunas pequeñas diferencias. El lenguaje que la computadora puede comprender directamente se denomina lenguaje máquina. El lenguaje ensamblador es casi la misma cosa, pero necesita un paso adicional para que la traducción pueda ser entendida por la máquina. Si un compilador traduce el programa de alto nivel a algún lenguaje de bajo nivel, no es exactamente lenguaje máquina, se necesita, por tanto, una pequeña traducción adicional antes de ser ejecutado en la computadora, pero normalmente este proceso suele ser automático y no es problemático.

En esencia, un compilador es un programa que traduce un programa en lenguaje de alto nivel, tal como un programa de C/C++/Java, en un programa de un lenguaje más sencillo que la computadora puede comprender más o menos directamente.

La compilación y sus fases

La compilación es el proceso de traducción de programas fuente a programas objeto. El programa objeto obtenido de la compilación se traduce normalmente a código máquina.

Para conseguir el programa máquina real se debe utilizar un programa llamado *montador o enlazador (linker).* El proceso de montaje conduce a un programa en lenguaje máquina directamente ejecutable. Figura 1.4.3.

El proceso de ejecución de un programa escrito en un lenguaje de programación (por ejemplo, C) y mediante un compilador suele tener los siguientes pasos:

1. Escritura del *lenguaje máquina* con un *editor* (programa que permite a una computadora actuar de modo similar a una máquina de escribir electrónica) y guardarlo en un dispositivo de almacenamiento (por ejemplo, un disco).
2. Introducir el programa fuente en memoria.
3. *Compilar* el programa con el compilador C.
4. *Verificar y corregir errores de compilación* (listado de errores).
5. Obtención del programa *objeto.*
6. El enlazador *(linker)* obtiene el *programa ejecutable.*
7. Se ejecuta el programa y, si no existen errores, se tendrá la salida del programa.



Figura 1.4.3. Fases de la compilación

El proceso de ejecución es mostrado en las figuras 1.4.4 y 1.4.5.

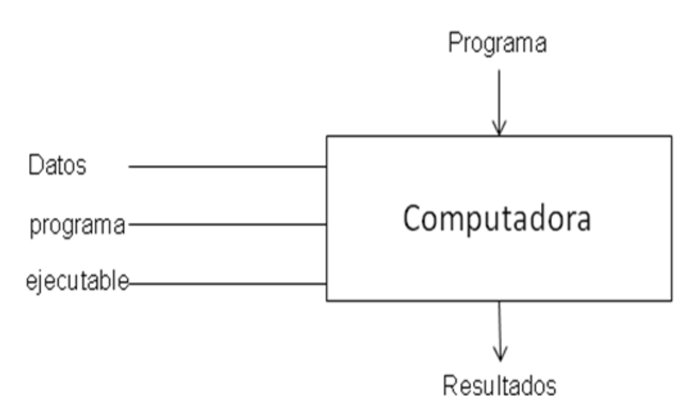


Figura 1.4.4. Ejecución de un programa

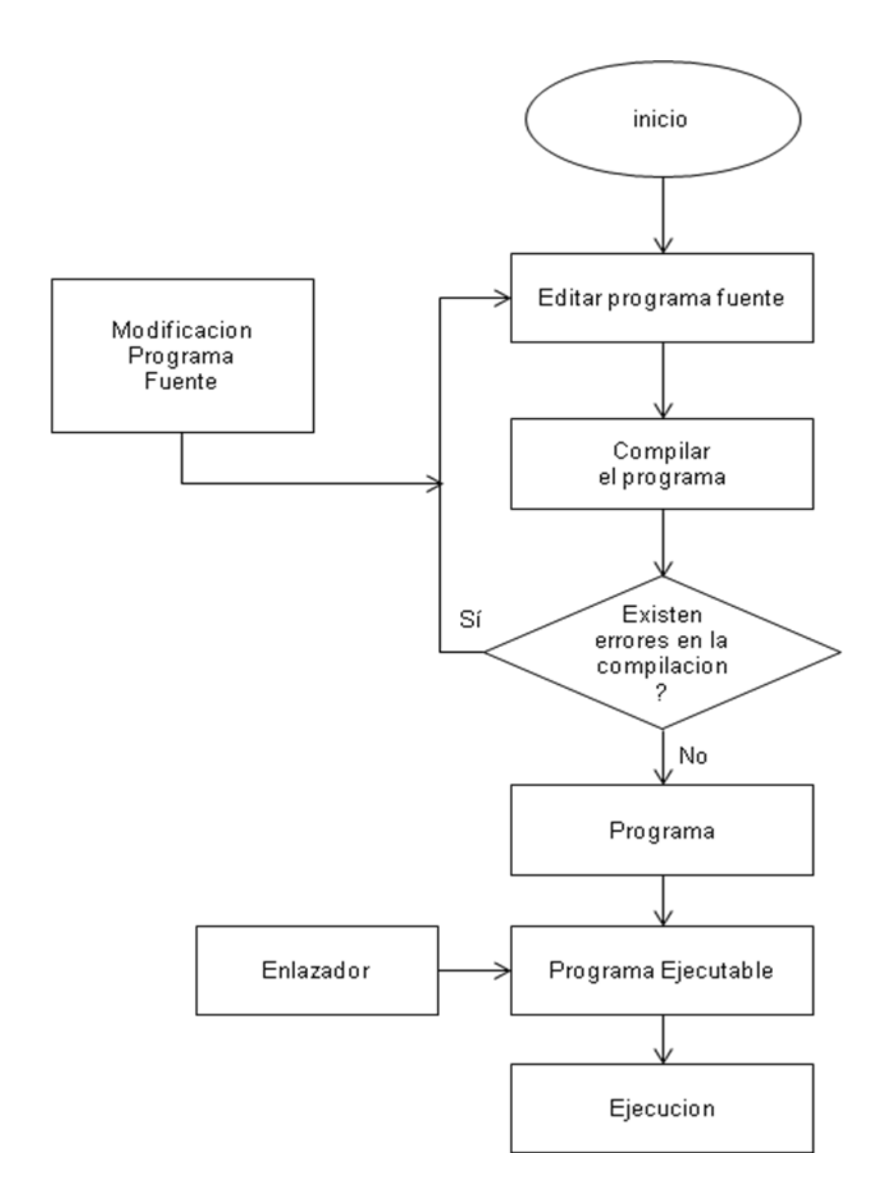


Figura 1.4.5. Fases de ejecución de un programa.

ACTIVIDADES DE APRENDIZAJE

1.- Realizar un cuadro sinóptico de los lenguajes de programación. Entrega impresa, mínimo 1 cuartilla. Especificar bibliografía consultada. Considerar limpieza y ortografía.

2.- Realizar un cuadro sinóptico de la clasificación de los lenguajes de programación. Entrega impresa, mínimo 1 cuartilla. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

3.- Realizar un cuadro sinóptico de los componentes de los lenguajes de programación, considerando sus características. Entrega impresa. Mínimo una cuartilla. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

4.- Realizar una síntesis de los lenguajes de programación. Entrega impresa, mínimo tres cuartillas. Especificar bibliografía consultada. Considerar limpieza y ortografía.

1.5. INTRODUCCIÓN AL LENGUAJE C++

Objetivo

El alumno conocerá las características del entorno integrado de desarrollo del lenguaje C++, así como sus reglas de sintaxis.

Introducción al Lenguaje C++

Joyanes Aguilar hace referencia que C++ es heredero directo del lenguaje C que a su vez se deriva del lenguaje B.

El lenguaje de programación C fue desarrollado por *Denis Ritchie de AT&T Bell Laboratories* que se utilizó para escribir y mantener el sistema operativo UNIX (hasta que apareció C, el sistema operativo UNIX fue desarrollado por *Ken Thompson en AT&T Bell Laboratories* mediante un lenguaje ensamblador o en B).

C es un lenguaje de propósito general que se puede utilizar para escribir cualquier tipo de programa, pero su éxito y popularidad está especialmente relacionado con el sistema operativo UNIX (fue desarrollado como *lenguaje de programación de sistemas,* es decir, un lenguaje de programación para escribir *sistemas operativos y utilidades,* programas, del sistema). Los sistemas operativos son los programas que gestionan (administran) los *recursos de la computadora.* Ejemplos bien conocidos de sistemas operativos además de UNIX son MS/DOS, OS/2, MVS, Linux, Windows 95/98, Windows NT, Windows 2.000, OS Mac, etc.

Aunque C es un lenguaje muy potente, tiene dos características que lo hacen inapropiado como una introducción moderna a la programación. Primero, C requiere un nivel de sofisticación a sus usuarios que les obliga a un difícil aprendizaje a los programadores principiantes ya que es de comprensión difícil.

Segundo C, fue diseñado al principio de los setenta, y la naturaleza de la programación ha cambiado de modo significativo en la década de los ochenta y noventa.

Para subsanar estas “deficiencias” *Bjarne Stroustrup de AT&T Bell Laboratories* desarrolló C++ al principio de la década de los ochenta. Stroustrup diseñó C++ como un mejor C. En general, C estándar es un subconjunto de C++ y la mayoría de los programas C son también programas C++ (la afirmación inversa no es verdadera). Señala Joyanes Aguilar, que C además de añadir propiedades de C, presenta características y propiedades de *programación orientada a objetos,* que es una técnica de programación muy potente.

Se han presentado varias versiones de C++ y su evolución se estudió por Stroustrup *(The Design and evolution of C++, AWL, 1994).* Las características más notables que han ido incorporándose a C++ son: herencia múltiple, genericidad, plantillas, funciones virtuales, excepciones, etc.

C++ comenzó su proyecto de estandarización ante el comité ANSI y su primera referencia es *The Annotated* C++ *Reference Manual.* En diciembre de 1989 se reunió el comité X3J16 del ANSI por iniciativa de Hewlett Packard. En junio de 1991, la estandarización de ANSI pasó a formar parte de un esfuerzo de estandarización ISO.[9]

Estructura General de un Programa en C++

Un programa en C++ se compone de una o más funciones. Una de las funciones debe ser obligatoriamente *main.* Una función en C++ es un grupo de instrucciones que realizan una o más acciones. Así mismo, un programa contendrá una serie de directivas *#include* que permitirían incluir en el mismo, archivos de cabecera que a su vez constarán de funciones y datos predefinidos en ellos.

---

[9] JOYANES AGUILAR, Luis. Programación en C++. Algoritmos estructuras de datos y objetos. Mc Graw Hill. España. 2000. P. 31.

Un programa C++ puede incluir:

- Variables locales y variables globales

En los lenguajes de programación existen por lo general dos tipos de variables: *Locales y Globales.* El tipo de variable depende de la sección en la cual sea declarada. Para el Lenguaje C++ si una variable es declarada antes de la sección del main (cabecera principal del programa) la variable se define como global, es decir, esta variable puede ser utilizada en cualquier sección del programa, llámese funciones o sección principal. Observar Figura 1.5.1.

Una variable local se caracteriza porque ésta solo puede ser utilizada en la sección donde fue declarada, es decir, si una variable es declarada en la sección de una función, sólo en esa sección puede ser utilizada. Si por el contrario es declarada en la sección principal del programa, sólo puede ser utilizada en esa área. Observar Figura 1.5.2.

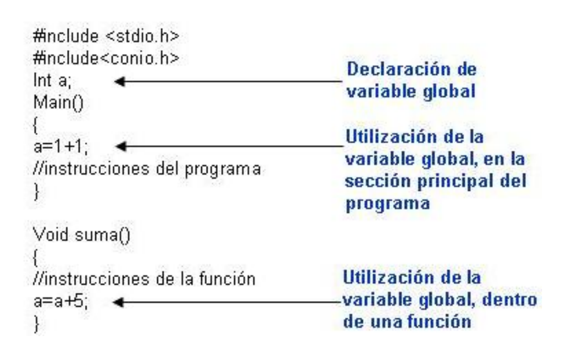


Figura 1.5.1 Declaración de Variable Global.

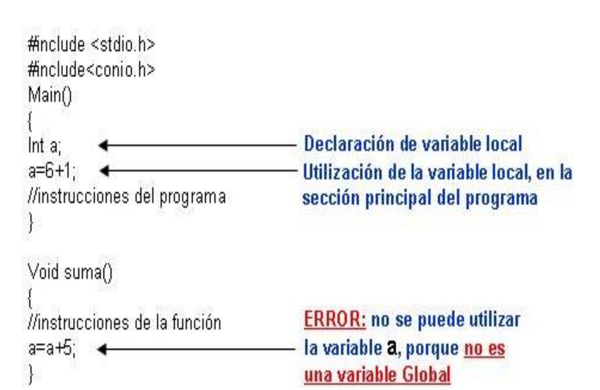


Figura 1.5.2 Declaración de Variable Local.

Constantes

En cualquier lenguaje de programación se emplea el concepto de constante, éstas a diferencia de las variables, conservan su valor durante toda la ejecución del programa. Las variables por el contrario modifican su valor durante la ejecución del programa.

Existen distintas tipos de constantes: numéricas y de texto.

Para declarar una constante en Turbo C++, se emplea la palabra reservada define antecedida del símbolo #:

#define nombre_constante valor_de_constante

Operadores aritméticos

El propósito de todo programa es dar solución a un problema mediante el empleo de la computadora.

Muchos de los problemas a solucionar involucran la realización de cálculos matemáticos, comparaciones entre cantidades.

Para realizar cálculos matemáticos, Turbo C++, proporciona los operadores aritméticos para las operaciones fundamentales, como: suma, resta, multiplicación, división, etc.

Los operadores aritméticos se clasifican en:

Los binarios:

+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo (resto)

Los unarios:

++	Incremento (suma 1)
- -	Decremento (resta 1)
-	Cambio de signo

Operadores de asignación

Los operadores de asignación tienen como objetivo, permitir el almacenamiento de determinado resultado en una variable; dichos cálculos previos pueden ser originados de extensas y complicadas fórmulas, por lo que estos operadores a parte de ser de asignación pretenden reducir la complejidad de las operaciones, a través de la simplificación de código.

Los operadores de asignación más comunes son:

=	Asignación simple
+=	Suma
-=	Resta
*=	Multiplicación
/=	División
%=	Módulo (resto)

Con estos operadores se puede programar, de forma más breve, expresiones del tipo:

n=n+3

Escribiendo:

n+=3

k=k*(x-2)

lo podemos sustituir por k*=x-2

ACTIVIDADES DE APRENDIZAJE

1.- Realizar una síntesis del lenguaje de programación de C++. Entrega impresa. Mínimo dos cuartillas. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

2.- Realizar un resumen de la historia del lenguaje de programación C++. Entrega impresa. Mínimo tres cuartillas. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

3.- Realizar práctica 1.

AUTOEVALUACIÓN

INSTRUCCIONES: Lee cuidadosamente y subraya la letra que corresponda a la palabra que complete la frase en cuestión.

1. Sus principales características es ser preciso, definido y finito, se habla de un: _________________

a) Algoritmo b) Análisis c) Diseño d) Prueba

2. Símbolo que representa el comienzo, "inicio" y el final, "fin" de un programa. Puede representar también una parada o interrupción programada que sea necesario realizar en un programa, denominado: ___________________.

a) Terminal b) Entrada/Salida c) Proceso d) Decisión

3. Símbolo que representa cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones matemáticas, de transferencia, etc., llamado: ______________

a) Terminal b) Entrada/Salida c) Proceso d) Decisión

4. En función del resultado de la comparación se seguirá uno de los diferentes caminos de acuerdo con dicho resultado, denominado: ________________.

a) Terminal b) Proceso c) Decisión d) Decisión múltiple

5. Indica el sentido de ejecución de las operaciones, denominado: ____________

a) Línea de flujo b) Línea conectora c) Conector d) Pantalla

INSTRUCCIONES: Lee cuidadosamente a cada pregunta y subraya la respuesta que corresponda.

6. Etapa en donde el problema de analiza teniendo presente la especificación de los requisitos dados por el cliente de la empresa o por la persona que se encarga del programa.

a) Diseño b) Codificación c) Análisis d) Documentación

7. Etapa en que la solución se escribe en la sintaxis del lenguaje de alto nivel (por ejemplo, C++) y se obtiene un programa.

a) Diseño b) Codificación c) Análisis d) Documentación

8. Etapa donde el programa se actualiza y modifica cada vez que sea necesario, de modo que se cumplan todas las necesidades de cambio de sus usuarios.

a) Depuración y mantenimiento b) Documentación.

c) Codificación (implementación). d) Diseño

9. Convierten las instrucciones escritas en lenguajes de programación en instrucciones escritas en lenguajes máquina (0 y 1, bits) que ésta pueda entender.

a) Comandos b) Pseudocódigo c) Traductores d) Lenguaje_máquina

10. Los operadores de asignación tienen como objetivo, permitir el almacenamiento de determinado resultado en una variable; dichos cálculos previos pueden ser originados de extensas y complicadas fórmulas, por lo que estos operadores a parte de ser de asignación pretenden reducir la complejidad de las operaciones, a través de la simplificación de código.

a) Operadores de asignación

b) Operadores aritméticos

c) Constantes

d) Variables

UNIDAD 2

EL ALGORITMO COMO ESTRATEGIA Y/O HERRAMIENTA PARA LA SOLUCIÓN DE PROBLEMAS.



http://www.ilanda.info/2008/07/la-estrategia-del-campen.html

OBJETIVO

El alumno será capaz de analizar y aplicar los conocimientos generales de las técnicas aplicadas, en la solución de algoritmos.

TEMARIO

2.1 Análisis para la solución de un problema

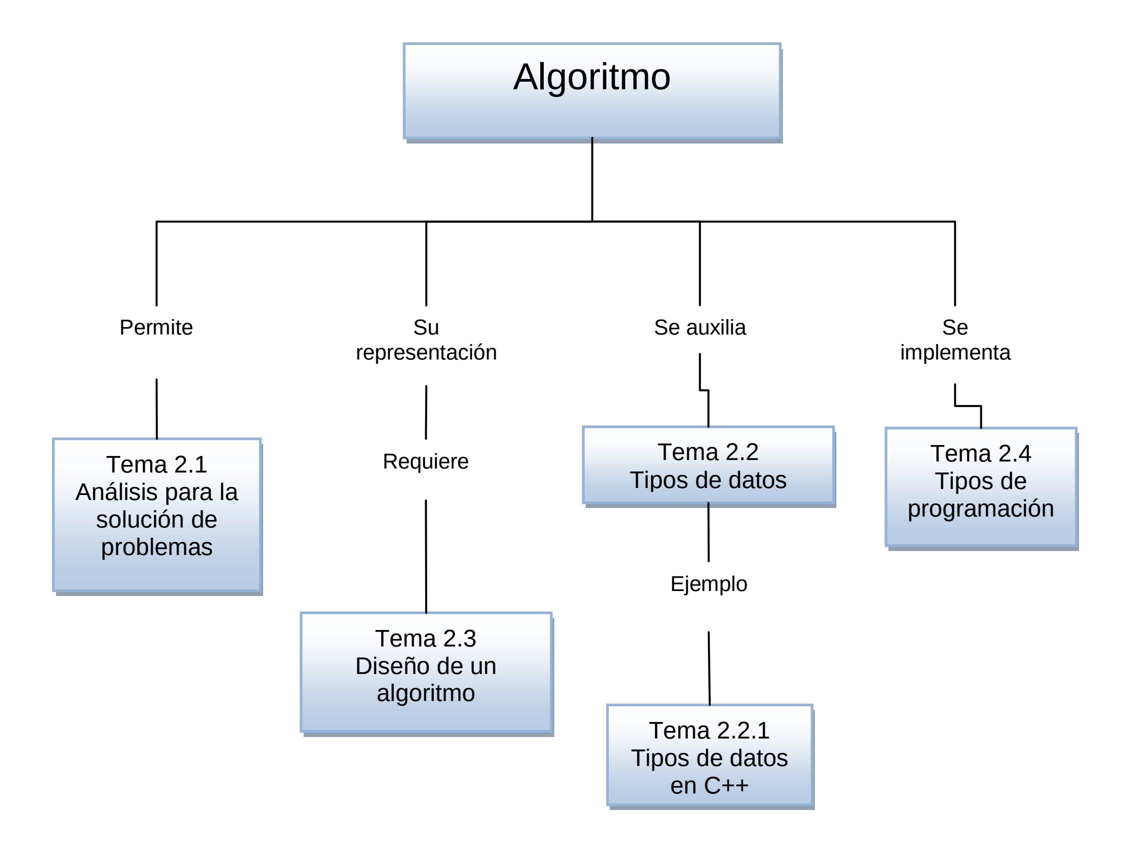
2.2 Tipos de datos

- 2.2.1 Tipos de datos en C++

2.3 Diseño de un algoritmos

2.4 Tipos de programación

MAPA CONCEPTUAL



INTRODUCCIÓN

Para desarrollar un algoritmo es importante conocer la metodología para tal propósito, así mismo identificar cuáles son los elementos primordiales de los que se conforma.

Todo algoritmo debe ser preciso, finito y definido por lo que su correcto diseño es vital para lograr tales características.

En el planteamiento de solución de problemas según la naturaleza del problema, existe la necesidad de manipular datos, en aquellas situaciones de problemas matemáticos, financieros, etc. Por lo que resulta trascendente conocer los tipos de datos que son representables en los algoritmos, y que guardan estrecha relación con los tipos de datos que emplean los lenguajes de programación actuales.

2.1. ANÁLISIS PARA LA SOLUCIÓN DE UN PROBLEMA

Objetivo

El alumno podrá explicar la importancia del análisis en la solución de un problema.

Análisis del problema[10]

En su libro de *fundamentos de programación*, Joyanes Aguilar, señala que la primera fase en la resolución de un problema con computadora es el *análisis del problema*. Esta fase requiere una clara definición, donde se contemple exactamente lo que debe hacer el programa y el resultado o solución deseada.

Debido a que se busca una solución por computadora, se precisan especificaciones detalladas de entrada y salida. Esto se plantea en la siguiente figura.



Figura 2.1. Análisis del problema.

Para poder definir bien un problema es conveniente responder a las siguientes preguntas:

[10] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. P.41.

- ¿Qué entradas se requieren? (tipo y cantidad)
- ¿Cuál es la salida deseada? (tipo y cantidad)
- ¿Qué método produce la salida deseada?

Ejemplo. Se desea calcular el total a pagar, en una venta normal en una papelería, proporcionando el precio unitario de un producto, así como el número de total de productos a comprar, además de aplicar el 15% de IVA.

Importe = precio_unitario*total de productos

IVA = importe*0.15

Total = importe+IVA

Entrada

- Precio unitario de producto
- Total de productos a comprar

Salida

- Total a pagar

Proceso

- Cálculo del Importe
- Cálculo del IVA
- Cálculo del total a pagar

ACTIVIDADES DE APRENDIZAJE

1.- Realizar avance de proyecto. Dependiendo del tipo de proyecto, el catedrático solicita los contenidos apropiados para este avance. La entrega será impresa. Considerar ortografía y limpieza.

2.- Realizar una síntesis de las fases para la resolución de problemas. Mínimo dos cuartillas. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

3.- Realizar un ejercicio donde se plantee las alternativas de solución para la problemática de cambio de llanta de un automóvil, señalando las acciones correspondientes de cada alternativa. Mínimo una cuartilla. Entrega impresa. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

4.- Realizar un ejercicio donde se plantee las alternativas de solución para la problemática del cálculo del área de un círculo. Mínimo una cuartilla. Entrega impresa. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

5.- Realizar un ejercicio donde se plantee las alternativas de solución para la problemática del cálculo del área de una esfera, un círculo, un cuadrado, de un cilindro, y de un rectángulo. Mínimo cuatro cuartillas. Entrega impresa. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

2.2. TIPOS DE DATOS

Objetivo

El alumno será capaz de explicar la importancia de los tipos de datos, en los algoritmos y programas.

Tipos de datos[11]

Con respecto a este tema, Joyanes Aguilar, considera que el principal objetivo en toda computadora es el manejo de la información o datos. Éstos pueden ser cifras de cualquier naturaleza, por ejemplo, de una boleta de calificaciones. Joyanes Aguilar define a un *dato* como la expresión general que describe los objetos con los cuales opera una computadora. La mayoría de las computadoras pueden trabajar con varios tipos (modos) de datos.

Señala que la acción de las instrucciones ejecutables de las computadoras se refleja en cambios en los valores de las partidas de datos. Los datos de entrada se transforman por el programa, después de las etapas intermedias, en datos de salida.

En el proceso de resolución de problemas el diseño de la estructura de datos es tan importante como el diseño del algoritmo y del programa que se basa en el mismo.

Existen dos tipos de datos: *simples* (sin estructura) y *compuestos* (estructurados).

Los distintos tipos de datos son representados en diferentes formas en la computadora. A nivel de máquina, un dato es un conjunto o secuencia de bits

[11] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p.90.

(dígitos 0 ó 1). Los lenguajes de alto nivel ignoran los detalles de la representación interna. Los tipos de datos simples son los siguientes:

Numéricos (*integer, real)*

Lógicos (*boolean*)

Carácter (*char, string*)

Datos numéricos

El tipo numérico es el conjunto de los valores. Estos pueden representarse en dos formas distintas:

- Tipo numérico *entero (integer).*
- Tipo numérico *real (real).*

Enteros: el tipo entero es un subconjunto finito de los números enteros. Los enteros son números completos, no poseen componentes fraccionarios o decimales y pueden ser negativos o positivos. Ejemplos de números enteros son:

10	16
-24	5
50	27
2009	26

Reales: el tipo real consiste en un subconjunto de los números reales, los cuales siempre tienen un punto decimal y pueden ser positivos o negativos. Un número real consta de un número y una parte decimal. Los siguientes ejemplos son números reales:

-5.7	3.1416
12.5	2009.03
9.10	3.10

Datos lógicos (booleanos)

El tipo *lógico*, señala Joyanes Aguilar, es también conocido como *booleano*, el cual es un dato que sólo puede tomar uno de dos valores:

Cierto o verdadero (true) y falso (false)

Este tipo de dato se utiliza para representar las alternativas *(si/no)* a determinadas condiciones. Por ejemplo, cuando se solicita si un valor entero es par, la respuesta será verdadera o falsa, según sea par o impar.

Datos tipo carácter y tipo cadena[12]

El tipo *carácter* es el conjunto finito y ordenado de caracteres que la computadora reconoce. Un dato tipo carácter contiene un solo carácter. Los caracteres que reconocen las diferentes computadoras no son estándar, sin embargo, la mayoría reconoce los siguientes caracteres alfabéticos y numéricos.

- Caracteres alfabéticos (A, B, C,…, Z) (a, b, c,…, z)
- Caracteres numéricos (1,2,3,…, 9, 0)
- Caracteres especiales (+ ,- ,* ,/ ,^ ,. , ; , <, >, $, …)

Una *cadena (string)* de caracteres es una sucesión que se encuentran delimitados por una comilla (apóstrofo) o dobles comillas, según el tipo de lenguaje de programación. La *longitud* de una cadena de caracteres es el número de ellos comprendidos entre los separadores o limitadores. Algunos lenguajes tienen datos tipo *cadena.*

‘hola saludos’
’10 de marzo de 2009’
‘Análisis de algoritmos’

[12] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p.92.

2.2.1 TIPOS DE DATOS EN C++[13]

Con respecto a este tema, Joyanes Aguilar, señala que C++ no soporta un gran número de tipos de datos predefinidos, pero tiene la capacidad para crear sus propios tipos de datos. Todos los tipos de datos simples o básicos de C++ son, generalmente, números. Los tres tipos de datos básicos son:

- Enteros
- Números de coma flotante (*reales*)
- Caracteres.

La tabla 1 ilustra los principales tipos de datos básicos, sus tamaños en bytes y el rango de valores que puede almacenar.

Tipo	Ejemplo	Tamaño en bytes	Rango Mínimo Máximo
Char	'c'	1	0..255
Short	-15	2	-128..127
Int	1024	2	-32768..32767
unsigned int	42325	2	0..65535
Long	262144	4	-2147483648..2147483637
Float	10.5	4	$3.4*(10^{-38})$. $3.4*(10^{38})$
Double	0.00045	8	$1.7*(10^{-308})$. $1.7*(10^{308})$
long double	$1e^{-8}$	8	$1.7*(10^{-308})$. $1.7*(10^{308})$

Tabla 1. Tipos de datos simples de C++.

Los tipos de datos fundamentales en C++ son:

- **Enteros** (números completos y sus negativos): de tipo int.
- **Variantes de enteros:** tipos short, long y unsigned.

[13] JOYANES AGUILAR, Luis. Programación en C++. Algoritmos, Estructuras de datos y objetos. Mc Graw Hill. España. 2000. p.50.

- **Reales:** números decimales: tipos float, doublé o long double.
- **Caracteres:** letras, dígitos, símbolos y signos de puntuación.

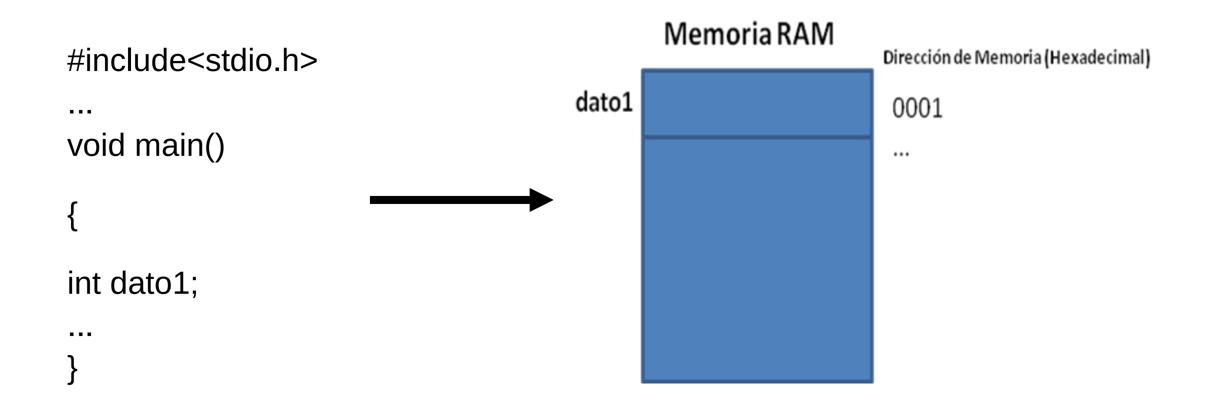
La característica común que guardan los datos de tipo *float y double* es que ambos permiten almacenar cifras decimales, así como enteras, pero se diferencian en que el último tiene un mayor rango de almacenamiento.

Estos tipos de datos son ampliamente utilizados para procesar cantidades muy grandes o muy pequeñas en el orden de los decimales.

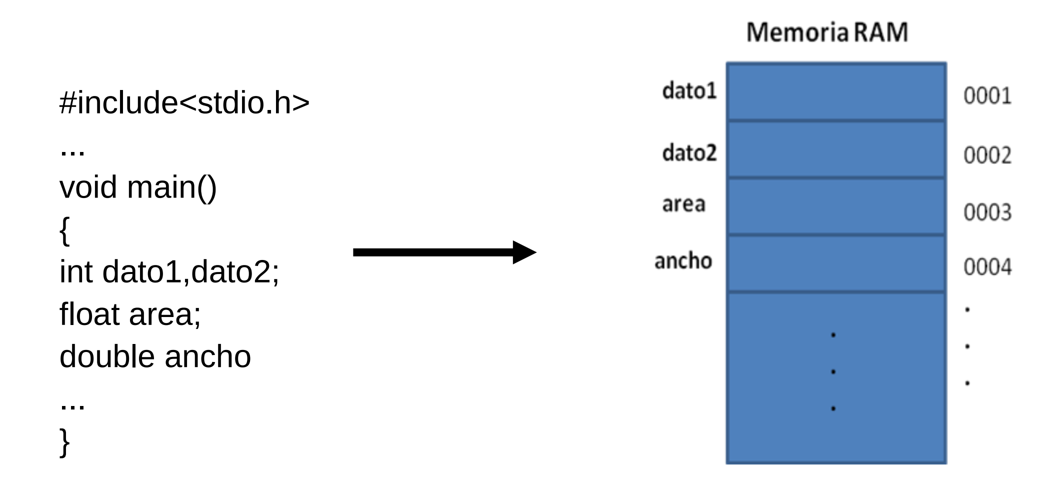
Si deseamos, por ejemplo, solicitar una calificación, el tipo de variable a declarar sería de tipo flotante (*float*), es decir, declarar una variable de este tipo tiene la característica de poder almacenar cifras decimales, o también sólo enteros.

A diferencia de una variable *int*, que solo almacena cifras decimales.
Explicación a nivel de memoria RAM, con respecto a la declaración de variables.
Cuando se declara una variable de cierto tipo de dato, y se ejecuta el programa esta variable ocupa un lugar en la Memoria RAM (Memoria de Acceso Aleatorio), este espacio que ocupa depende del tipo de dato de la variable, por eso que es importante tener conocimiento de cual es el tamaño del tipo de dato.

- Por ejemplo, una variable de tipo *int* ocupa 2 bytes en memoria.
- Si declaramos una variable de este tipo en un programa:



Nota: Cada vez que declaras una variable en un programa y este es ejecutado, por cada variable a nivel memoria, ocupa un espacio dependiendo del tipo de dato.



Ejemplo de implementación de tipos de datos flotante.

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
        float a,b,h;
        clrscr();  //función para limpiar pantalla;
        cout<<"\nCalculando el área de un rectángulo";
        cout<<"\nIntroduce el valor de la base:";
        cin>>b;
        cout<<"\nIntroduce el valor de la altura:";
        cin>>h;
        a=b*h;  //formula para calcular el área del rectángulo
        cout<<"\nEl valor de la altura es:"<<a;
        getch();
}
```

Ejemplo de tipos de dato entero y carácter.

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
	char nombre[50];
	int edad,anyo,natalicio;
	clrscr();  //función para limpiar pantalla;
	cout<<“\nIntroduce tu nombre”;
	gets(nombre);
	cout<<“\nIntroduce el año actual:”;
	cin>>anyo;
	cout<<“\nIntroduce el año de tu nacimiento:”;
	cin>>natalicio;
	edad=anyo-natalicio;
	cout<<“\nTu edad es:”<<edad;
	getch();
}
```

ACTIVIDADES DE APRENDIZAJE

1.- Realizar un cuadro sinóptico de los tipos de datos más comunes. Mínimo una cuartilla. Entrega impreso. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

2.- Realizar un cuadro sinóptico de los tipos de datos de C++ más comunes ejemplificando cada uno de ellos. Mínimo dos cuartillas. Entrega impresa. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

3.- Realizar práctica 2.

4.- Realizar un resumen de los tipos de datos más comunes, señalar ejemplos de aplicación de cada tipo de dato. Mínimo tres cuartillas. Entrega impreso. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

5.- Realizar un resumen de los tipos de datos de C++ más comunes, ejemplificando cada uno de ellos. Señalar el espacio que ocupan en memoria, especificar el rango de datos que permiten almacenar. Mínimo tres cuartillas. Entrega impresa. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

2.3. DISEÑO DE UN ALGORITMO

Objetivo

El participante obtendrá los conocimientos para explicar las características de los elementos de un algoritmo. Resolverá algoritmos matemáticos, por ejemplo, de cálculo de perímetros, de áreas, cálculo de promedios.

Diseño del algoritmo[14]

Joyanes Aguilar, señala que en la etapa de análisis del proceso de programación se determina, ¿qué hace el programa? En la etapa de diseño se determina cómo hace el programa la tarea solicitada. Los métodos más eficaces para el proceso de diseño se basan en el conocido por *divide* y *vencerás.* Es decir, la resolución de un problema complejo se divide en subproblemas y después dividir éstos en otros de nivel más bajo, hasta que pueda ser *implementada* una solución en la computadora. Este método se conoce técnicamente como *diseño descendente (top-down)* o *modular.* El proceso de romper el problema en cada etapa y expresar cada paso en forma más detallada se denomina *refinamiento sucesivo.*

Cada subprograma es resuelto mediante un *módulo (subprograma)* que tiene un solo punto de entrada y un solo punto de salida.

Cualquier programa bien diseñado consta de *un programa principal* (el módulo de nivel más alto) que llama a subprogramas (módulos de nivel más bajo) que a su vez pueden llamar a otros subprogramas. Los programas estructurados de esta forma se dice que tienen un *diseño modular* y el método de romper el programa en módulos más pequeños se llama *programación modular.* Los módulos pueden ser planeados, codificados, comprobados y depurados

[14] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p.42.

independientemente (incluso por diferentes programadores) y a continuación combinarlos entre sí. El proceso implica la ejecución de los siguientes pasos hasta que el programa se termina:

1.- Programar un módulo.
2.- Comprobar el módulo.
3.- Si es necesario, depurar el módulo.
4.- Combinar el módulo con los módulos anteriores.

Joyanes Aguilar, menciona que el proceso que convierte los resultados del análisis del problema en un diseño modular con refinamientos sucesivos que permitan una posterior traducción a un lenguaje se denomina *diseño del algoritmo.*

Algo muy importante que señala Joyanes Aguilar, se centra en que el diseño del algoritmo es independiente del lenguaje de programación en el que se vaya a codificar posteriormente.

Herramientas de programación[15]

En su libro de fundamentos de programación, Joyanes Aguilar, señala que las dos herramientas más utilizadas comúnmente para diseñar algoritmos son: *diagramas de flujo y pseudocódigos.*

Diagramas de flujo

Un diagrama de flujo (flowchart) es una representación gráfica de un algoritmo. Los símbolos utilizados en estos diagramas, descritos en lecturas anteriores, han sido normalizados por el Instituto Norteamericano de Normalización (ANSI).

Pseudocódigo

El pseudocódigo es una herramienta de programación en la que las instrucciones se escriben en palabras similares al inglés o español, que facilitan

---

[15] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p.41.

tanto la escritura como la lectura de programas. En esencia, el pseudocódigo se puede definir como *un lenguaje de especificaciones de algoritmos.*

Aunque no existen reglas para escritura del pseudocódigo es español, se utilizan palabras reservadas básicas, estas palabras son traducción libre de palabras reservadas de lenguajes como C, Pascal, etc.

Ejemplo:

Se desea calcular el total a pagar, en una venta normal en una papelería, proporcionando el precio unitario de un producto, así como el número de total de productos a comprar, además de aplicar un IVA del 15%.

Pseudocódigo:

Calculando total a pagar

Introducir precio unitario

Numero total de productos a comprar

Calcular importe

Calcular importe más IVA

Imprimir total a pagar

Ejemplo: Algoritmo para calcular el área de un rectángulo.

1.- Inicio

2.- Obtener el valor del largo, denominado LARGO

3.- Obtener el valor del ancho, denominado ANCHO

4.- Multiplicar el valor de ANCHO por LARGO, llamar al resultado: AREA

5.- El área del rectángulo es: AREA.

6.- Fin

Ejemplo: Algoritmo para realizar la conversión de minutos en segundos.

1.- Inicio

2.- Obtener el número de minutos a convertir, denominado MINUTOS.

3.- Multiplicar MINUTOS por 60, llamar al resultado MINUTOS

4.- Visualizar el resultado en minutos: MINUTOS.

5.- Fin

Ejemplo: Algoritmo para calcular el promedio de dos materias.

1.- Inicio

2.- Leer la calificación de la materia 1, denominada MATERIA1

3.- Leer la calificación de la materia 2, denominada MATERIA2

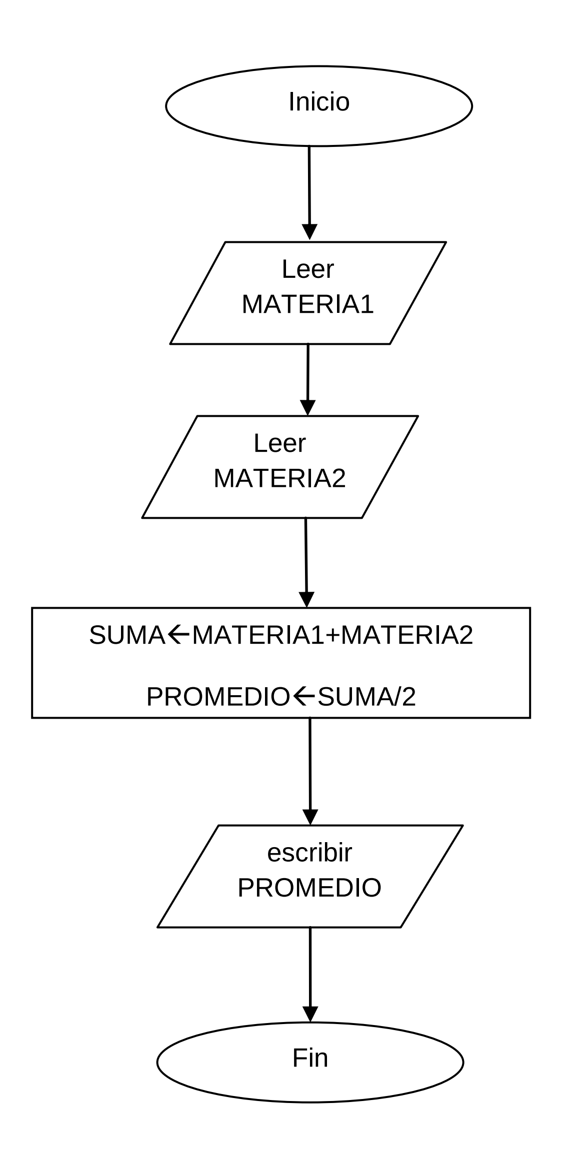
4.- Sumar MATERIA1 más MATERIA2, el resultado denominarlo: SUMA

5.- Dividir a SUMA entre 2, el resultado denominarlo PROMEDIO

6.- Visualizar el promedio de las materias, imprimir PROMEDIO.

7.- Fin

Ejemplo: Diagrama de flujo, que permite calcular el promedio de dos materias.



ACTIVIDADES DE APRENDIZAJE

1.- Realizar un algoritmo que permita calcular el área de un rectángulo. Entrega impresa. Considerar limpieza y ortografía. Mínimo una cuartilla.

2.- Realizar un algoritmo que permita calcular el área de un triángulo. Entrega impresa. Considerar limpieza y ortografía. Mínimo una cuartilla.

3.- Realizar un algoritmo que permita calcular la edad actual de una persona, solicitando su año de nacimiento. Entrega impresa. Considerar limpieza y ortografía. Mínimo una cuartilla.

4.- Realizar un algoritmo que permita calcular el año de nacimiento de una persona, solicitando su edad. Entrega impresa. Considerar limpieza y ortografía. Mínimo una cuartilla.

5.- Realizar un algoritmo que permita calcular el promedio de un alumno, el cual tiene 10 materias y cuya calificación se solicita previamente. Entrega impresa. Considerar limpieza y ortografía. Mínimo una cuartilla.

6.- Realizar un algoritmo que permita calcular la velocidad que emplea un móvil, considerando que velocidad=distancia/tiempo. Entrega impresa. Considerar limpieza y ortografía. Mínimo una cuartilla.

7.- Realizar un algoritmo que permita calcular la velocidad que emplea un móvil, considerando que velocidad=distancia/tiempo. Entrega impresa. Considerar limpieza y ortografía. Mínimo una cuartilla.

8.- Realizar un algoritmo que permita calcular la distancia que emplea un móvil, considerando que tiempo=distancia/velocidad. Entrega impresa. Considerar limpieza y ortografía. Mínimo una cuartilla.

9.- Realizar un algoritmo que permita calcular el área de un círculo. Entrega impresa. Considerar limpieza y ortografía. Mínimo una cuartilla.

10.- Realizar práctica 3.

11.- Realizar un algoritmo que permita calcular el área de un cilindro. Entrega impresa. Considerar limpieza y ortografía. Mínimo una cuartilla.

12.- Realizar un algoritmo que permita calcular el número de días transcurridos de una persona desde su año de nacimiento. Entrega impresa. Considerar limpieza y ortografía. Mínimo una cuartilla.

13.- Realizar un algoritmo que permita calcular el promedio de estaturas, de 10 alumnos cuyos datos se solicitan previamente. Entrega impresa. Considerar limpieza y ortografía. Mínimo una cuartilla.

14.- Realizar un algoritmo que permita calcular el número de milímetros contenidos en una determinada cantidad de metros solicitados previamente. Entrega impresa. Considerar limpieza y ortografía. Mínimo una cuartilla.

15.- Realizar un algoritmo que permita calcular el número de segundos contenidos en una determinada cantidad de horas solicitadas previamente. Entrega impresa. Considerar limpieza y ortografía.

16.- Realizar un algoritmo que permita calcular el número de milímetros contenidos en una cantidad de kilómetros solicitados previamente. Entrega impresa. Considerar limpieza y ortografía.

17.- Realizar un algoritmo que permita calcular el número de segundos contenidos en un determinado número de días solicitados previamente. Entrega impresa. Considerar limpieza y ortografía.

2.4. TIPOS DE PROGRAMACIÓN

Objetivo

El alumno conocerá las características de los tipos de programación e identificará su importancia.

Tipos de programación[16]

Norton señala que hasta los años setenta, relativamente existía poca estructura en la escritura de códigos por parte de los programadores. Por ejemplo, los programadores con frecuencia usaban **instrucciones goto** para saltar otras partes de un programa. El problema que presenta esta instrucción es identificar cómo procede el flujo de control del programa después del salto.

Programación Estructurada

Los investigadores en los años sesenta demostraron que los programas podían escribirse con tres estructuras de control:

La estructura de la secuencia define el flujo de control automático en un programa, la cual se construye en lenguaje de programación. Como resultado, una computadora ejecuta líneas de código en el orden en el cual están escritas. La figura 2.4.1 muestra un diagrama de este flujo de secuencia.

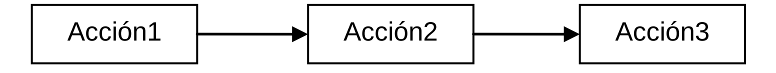


Figura 2.4.1. Estructura de la secuencia

Los comandos en los rectángulos representan tres líneas secuenciales de código. El control del programa fluye de la línea anterior de código a la siguiente línea. Los comandos están escritos en pseudocódigo, que es un lenguaje

[16] NORTON, Peter. Introducción a la Computación. Mc Graw Hill. México. 3ra edición. 2000. p. 423.

informal que los programadores usan mientras están trabajando con la lógica de un programa. Después de la secuencia de comandos es desarrollada, los programadores traducen el pseudocódigo a un lenguaje específico de cómputo.

Las estructuras de selección se construyen con base en una *declaración condicional.* Si ésta es verdadera, ciertas líneas de código son ejecutadas. Si por el contrario, es falsa, esas líneas de código no son ejecutadas. Las estructuras de selección más comunes son: if-then e if-else (llamada algunas veces como if-then-else. Las figuras 2.4.2 y 2.4.3 ilustran estos tipos de estructuras.

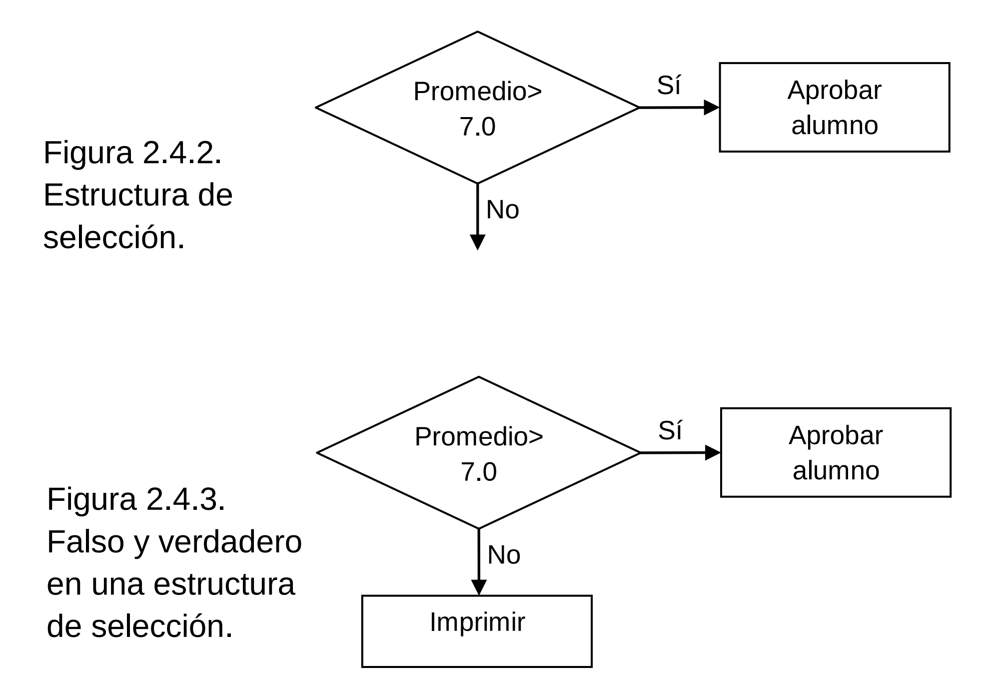


Figura 2.4.2. Estructura de selección.

Figura 2.4.3. Falso y verdadero en una estructura de selección.

Las *estructuras de repetición (o de ciclo)*[17] también se construyen con base en instrucciones condicionales. Si la condición es verdadera entonces un bloque de uno o más comandos se repite hasta que la condición es falsa. La computadora primero valida la condición y, si es verdadera, ejecuta el bloque de

[17] NORTON, Peter. Introducción a la Computación. Mc Graw Hill. México. 3ra edición. 2000. p. 424.

comando una vez. Entonces prueba la condición otra vez. Si aún es verdadera, el bloque de comando se repite. Debido a este funcionamiento cíclico, las estructuras de repetición son llamadas también **ciclos**. Tres estructuras cíclicas comunes son: For-Next, While y Do-While. Las figuras 2.4.4, 2.4.5 y 2.4.6 ilustran estas tres estructuras cíclicas.

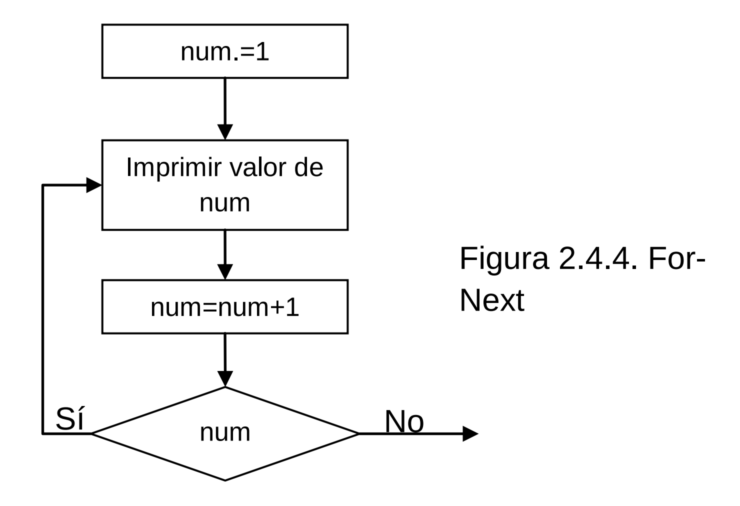


Figura 2.4.4. For-Next

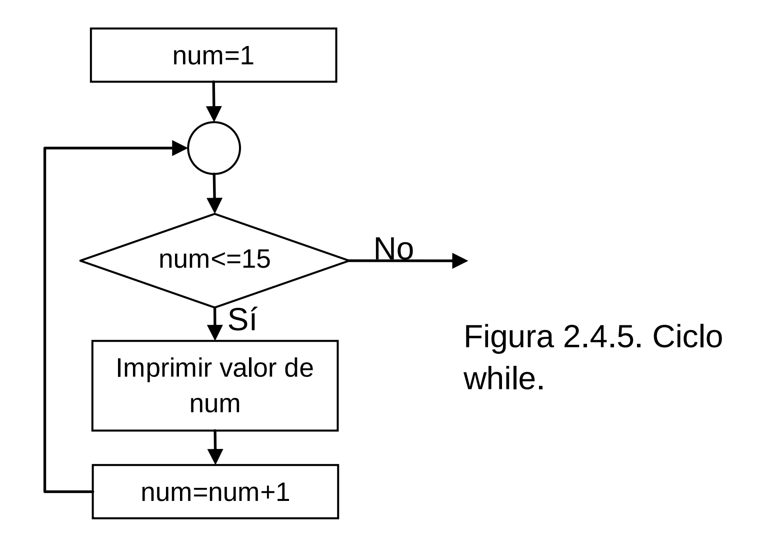


Figura 2.4.5. Ciclo while.

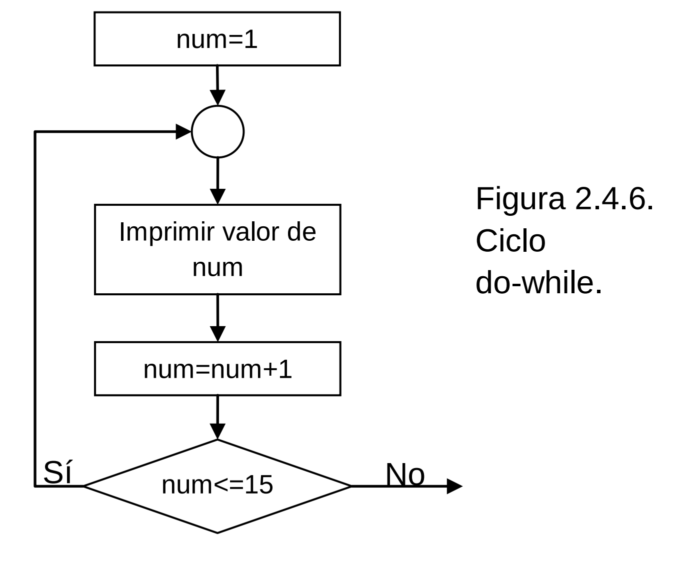


Figura 2.4.6. Ciclo do-while.

Programación Orientada a Objetos.

Norton, señala que los programadores afirman que una orientación a objetos es una manera natural de pensar acerca del mundo, ya que es una manera intuitiva para moldear el mundo, los programas se vuelven más simples, la programación más rápida, y el problema de mantenimiento al programa disminuye.[18]

Joyanes, afirma que el paradigma orientado a objetos nació en 1969 de la mano del doctor noruego *Kristin Nygaard* que intentando escribir un programa de computadora que describiera el movimiento de los barcos a través de un fiordo (Golfo, estrecho en el mar), halló que era muy difícil simular las mareas, los movimientos de los barcos y las formas de la línea de la costa con los métodos

[18] NORTON, Peter. Introducción a la Computación. Mc Graw Hill. México. 3ra edición. 2000. p. 424.

de programación existentes en ese momento. Descubrió que los elementos del entorno que trataba de modelar, barcos, mareas y línea de la costa de los fiordos y las acciones que cada elemento podía ejecutar, constituían unas relaciones que eran más difíciles de manejar.

Concluye que las tecnologías orientadas a objetos, han evolucionado mucho pero mantiene la razón de ser del paradigma: *combinación de la descripción de los elementos en un entorno de proceso de datos con las acciones ejecutadas por esos elementos.* Las clases y objetos como instancias o ejemplares de ellas son los elementos claves sobre los que se articula la orientación a objetos.

¿Qué son los objetos?[19]

Las personas en el mundo real, identifican los objetos como cosas que pueden ser percibidas por los cinco sentidos. Los objetos tienen propiedades específicas, tales como posición, tamaño, color, forma, textura, etc., que definen su estado, señala Joyanes. Agrega que los objetos tienen ciertos comportamientos que los hacen diferentes de otros objetos.

Cualquier programa orientado a objetos puede manejar muchos objetos. Por ejemplo, un programa que lleva el inventario de un almacén de ventas al por menor, utiliza un objeto de cada producto manipulado en el almacén. El programa manipula los mismos datos de cada objeto, incluyendo el número de producto, descripción del producto, precio, número de artículos del *stock* y el momento de nuevos pedidos.

Cada objeto conoce también cómo ejecutar acciones con sus propios datos, señala Joyanes. El objeto producto del programa de inventario, por ejemplo, conoce cómo crearse a sí mismo y establecer los valores iniciales de todos sus datos, cómo identificar sus datos y cómo evaluar si hay artículos suficientes en el *stock* para cumplir una petición de compra. En esencia, la cosa más

[19] JOYANES AGUILAR, Luis. Programación en C++. Algoritmos, Estructuras de datos y objetos. Mc Graw Hill. España. 2000. p.265.

importante de un objeto es reconocer que consta de datos, y las acciones que pueden ejecutar.

Joyanes enfatiza que un objeto en un programa de computadora no es algo que se pueda tocar. Cuando un programa se ejecuta, la mayoría de los objetos existen en memoria principal. Estos objetos se crean por un programa para su uso mientras el programa se está ejecutando. A menos que se guarden los datos de un objeto en un dispositivo de almacenamiento, el objeto se pierde cuando el programa termina (este objeto se llama *transitorio* para diferenciarlo del objeto *permanente* que se mantiene después de la terminación del programa).

¿Qué son las clases?[20]

Joyanes, define a una clase como un tipo de objeto especificado por el usuario. Las clases son los bloques de construcción fundamentales de los programas orientados a objetos. Una clase contiene la especificación de los datos que describen un objeto junto con la descripción de las acciones que un objeto conoce cómo ha de ejecutar. Estas acciones se conocen como *servicios, métodos o funciones, miembro.*

Antes de que un programa pueda crear objetos de cualquier clase, ésta debe ser *definida.* La definición de una clase significa que se debe dar a la misma un nombre, darle nombre a los elementos que almacenan sus datos y describir las funciones que realizarán las acciones consideradas en los objetos.

[20] JOYANES AGUILAR, Luis. Programación en C++. Algoritmos, Estructuras de datos y objetos. Mc Graw Hill. España. 2000. p.266.

ACTIVIDADES DE APRENDIZAJE

1.- Realizar un cuadro sinóptico de los tipos de programación. Especificar las características de cada uno de ellos (año de surgimiento, precursores, lenguajes de programación de ejemplo, características generales).entrega impresa. Mínimo 2 cuartillas. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

AUTOEVALUACIÓN

INSTRUCCIONES: Lee cuidadosamente y subraya la letra que corresponda a la palabra que complete la frase en cuestión.

1. Un ___________ es la expresión general que describe los objetos con los cuales opera una computadora

a) Campo b) Dato c) Diseño d) Registro

2. Es un tipo de dato que forma un subconjunto finito de los números enteros. Son números completos, no poseen componentes fraccionarios o decimales y pueden ser negativos o positivos.___________________

a) Entero b) Lógico c) Real d) Date

3. Tipo de dato, también conocido como booleano, el cual sólo puede tomar uno de dos valores: falso o verdadero. ____________________

a) Lógico b) Time c) Real d) Date

4. Tipo de dato que es un conjunto finito y ordenado de caracteres que la computadora reconoce.

a) Cadena b) Carácter c) Lógico d) Date

5. Es el número de caracteres comprendidos entre los separadores o limitadores.___________________

a) Cadena b) Longitud de una cadena

c) Longitud real d) Longitud booleana

INSTRUCCIONES: Lee cuidadosamente a cada pregunta y subraya la respuesta que corresponda.

6. Método donde la resolución de un problema complejo se divide en subproblemas y después dividir éstos en otros de nivel más bajo, hasta que pueda ser implementada una solución en la computadora.

a) Algorítmica b) Simple c) Modular d) Analítica

7. Instituto que normaliza los símbolos utilizados en los diagramas de flujo.

a) OSI b) ISO c) ANSI d) IEEE

8. Es un lenguaje informal que los programadores usan mientras están trabajando con la lógica de un programa.

a) C++ b) Pseudocódigo c) UNIX d) Máquina

9. Las sentencias: For-Next, While y Do-While son ejemplos de estructuras de tipo.

a) Secuencial b) Selección

c) Repetitivas d) Cíclicas

10. Cuando un objeto se pierde cuando el programa termina, recibe el nombre de.

a) Permanente b) Transitorio c) Simple d) Único

UNIDAD 3

ESTRUCTURAS BÁSICAS DE UN ALGORITMO.



www.skyscrapercity.com/showthread.php?p=31763124

OBJETIVO

El estudiante analizará las sentencias que permiten utilizar técnicas de solución en los algoritmos.

TEMARIO.

3.1 Estructuras secuenciales

3.2 Estructuras condicionales

3.3 Contadores

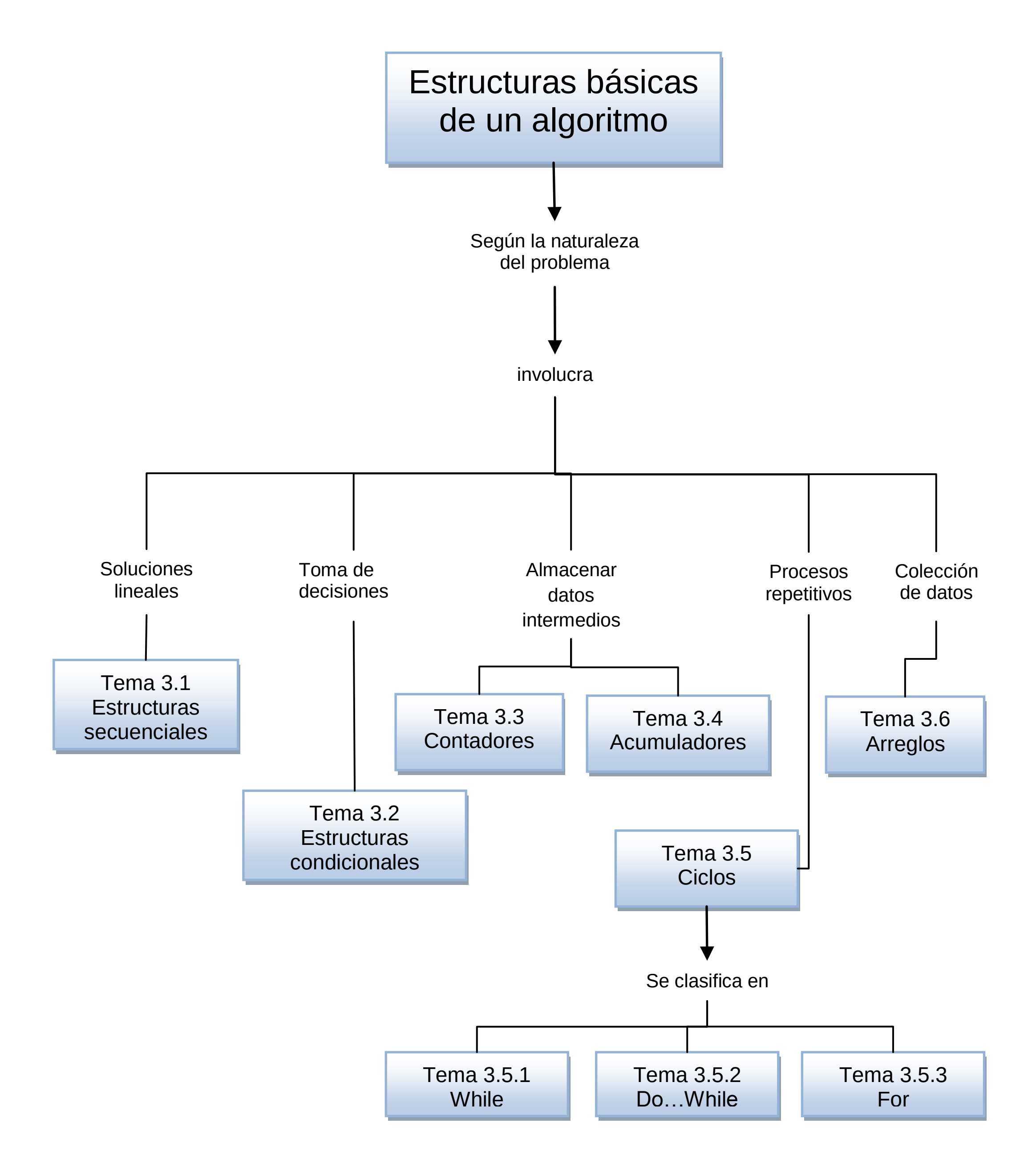
3.4 Acumuladores

3.5 Ciclos

- 3.5.1 While
- 3.5.2 Do…while
- 3.5.3 For

3.6 Arreglos

MAPA CONCEPTUAL



INTRODUCCIÓN

Una vez comprendida la importancia de un algoritmo como herramienta para la solución de problemas, es necesario conocer los elementos disponibles para desarrollar los algoritmos.

Dependiendo de la naturaleza del problema a analizar están disponibles: estructuras secuenciales para aquellas circunstancias en las que una simple secuencia lineal de acción no permite dar solución a un determinado problema. De la misma manera, cuando la situación amerita tomar decisiones dentro de los algoritmos es posible recurrir a las estructuras condicionales, las cuales tienen como principal característica la evaluación de una condición para realizar o no un determinado conjunto de acciones.

En ocasiones existen actividades y acciones que son repetitivas, por lo que los ciclos (estructuras repetitivas) presentan características que permiten su implementación en tales situaciones.

En el ámbito de manipulación de la información existe la necesidad de manipular grandes cantidades de datos, ya sea para organizarlos, ubicarlos y demás acciones; por lo que los arreglos son una necesidad indispensable para manipular conjuntos de datos del mismo tipo.

En esencia, todos los elementos descritos anteriormente son la base de los algoritmos, ya que a través de ellos es posible plantear las soluciones a los problemas.

3.1. ESTRUCTURAS SECUENCIALES

Objetivo

El alumno será capaz de explicar la importancia de las estructuras secuenciales en el diseño de algoritmos. Resolverá problemas básicos mediante diagramas de flujo.

Estructura secuencial[21]

Joyanes Aguilar, define a una estructura secuencial como aquella en la que una acción (instrucción) sigue a otra en serie. Señala que las tareas suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el final del proceso. La estructura secuencial tiene una entrada y una salida. Su representación gráfica se muestra en la figura 3.1.

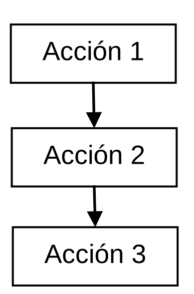


Figura 3.1. Estructura secuencial.

Ejemplo 1: Cálculo de la suma y producto de tres números.

La suma S de dos números es S=A+B+C y el producto P=A*B*C. El pseudocódigo y diagrama de flujo se muestra a continuación.

[21] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 132.

Pseudocódigo

inicio

Leer(A)

Leer(B)

Leer(C)

S←A+B+C

P←A*B*C

Escribir(S,P)

fin

Diagrama de flujo

inicio

Entero: A,B,C,S,P

Leer
A

Leer
B

Leer
C

S←A+B+C
P←A*B*C

escribir
S,P

fin

Ejemplo 2: Cálculo de la estatura promedio de 3 alumnos.

Pseudocódigo

Inicio

Leer(estatura1,estatura2,estatura3)

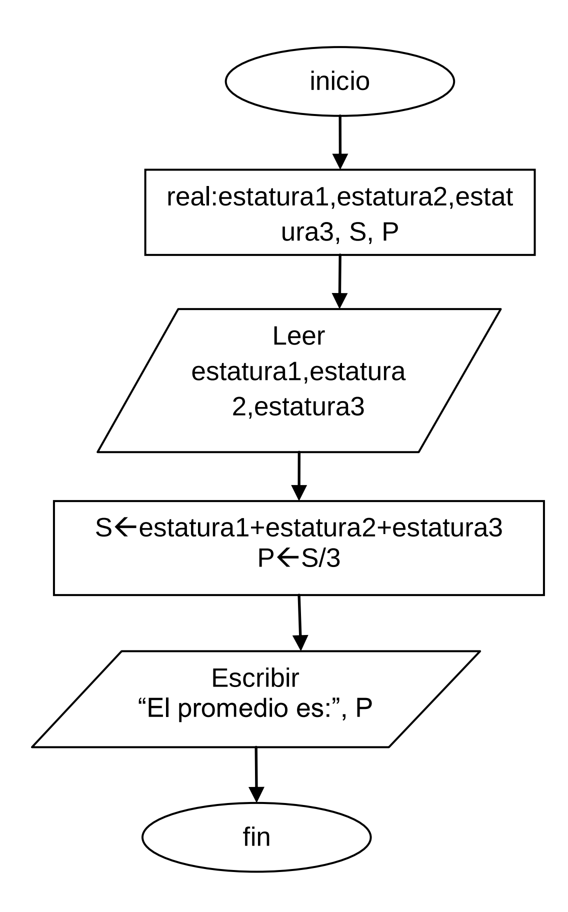
S←estatura1+estatura2+estatura3

P←S/3

Escribir (“El promedio es:”,P)

fin

Diagrama de flujo

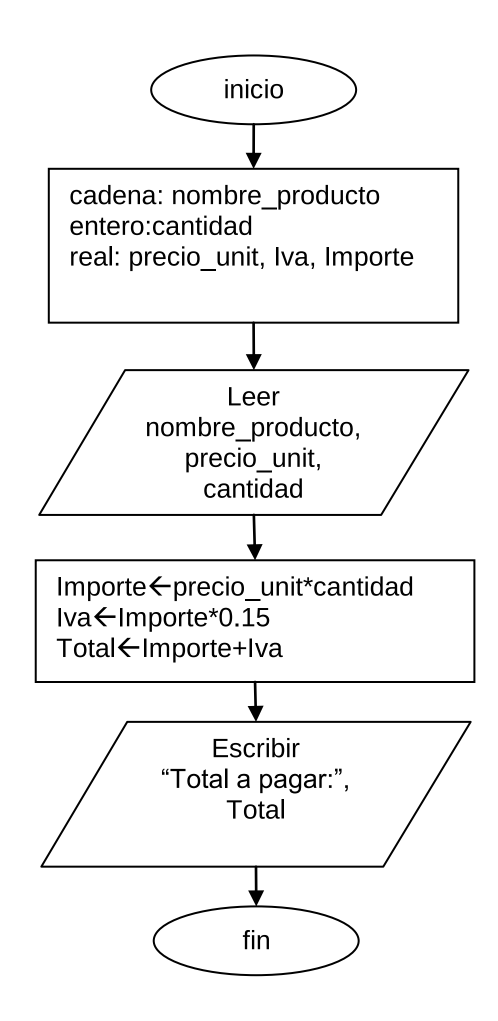


Ejemplo 3: Diagrama de flujo que representa el proceso de compra en una papelería, en donde se aplica un IVA del 15% en cualquier compra.

Pseudocódigo

```
Inicio
        Leer (nombre_producto)
        Leer (precio_unit)
        Leer (cantidad)
        Importe←precio_unit*cantidad
        Iva←Importe*0.15
        Total←Importe+Iva
        Escribir ("Total a pagar:",Total)
fin
```

Diagrama de flujo

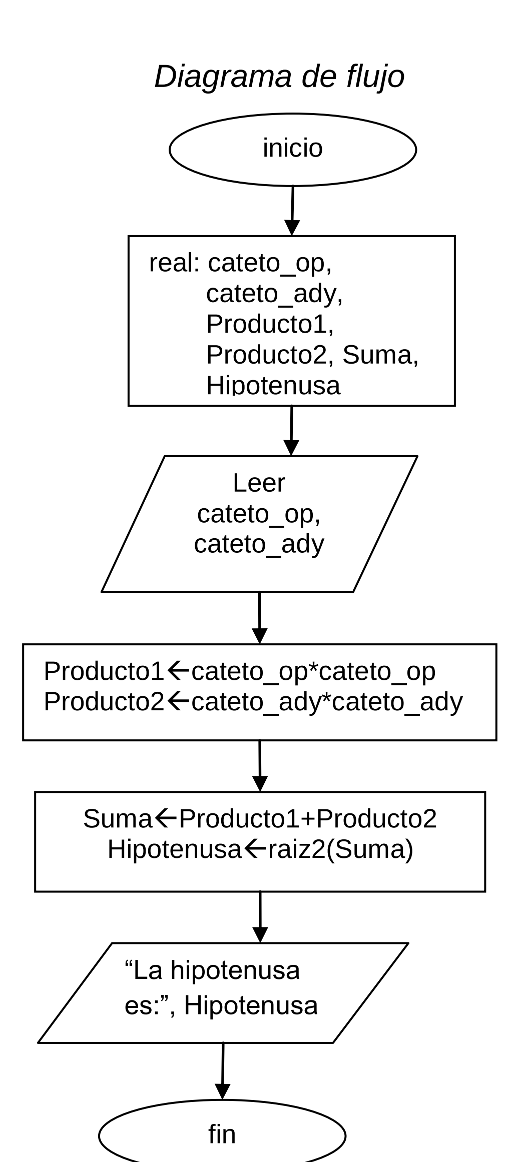


Ejemplo 4: Diagrama de flujo que permite calcular la hipotenusa de un triángulo rectángulo, conociendo el valor de los catetos.

Pseudocódigo

```
Inicio
        Leer (cateto_op, cateto_ady)
        Producto1←cateto_op*cateto_op
        Producto2←cateto_ady*cateto_ady
        Suma←Producto1+Producto2
        Hipotenusa←raíz2(Suma)
        Escribir ("La hipotenusa es:", Hipotenusa)
fin
```

Diagrama de flujo



ACTIVIDADES DE APRENDIZAJE

1.- Realizar un diagrama de flujo que permita calcular el área de un rectángulo. Entrega impresa. Considerar limpieza y ortografía. Mínimo en una cuartilla.

2.- Realizar un diagrama de flujo que permita calcular el área de un triángulo. Entrega impresa. Considerar limpieza y ortografía. Mínimo en una cuartilla.

3.- Realizar un diagrama de flujo que permita calcular la edad de una persona solicitando su año de nacimiento. Entrega impresa. Considerar limpieza y ortografía. Mínimo en una cuartilla.

4.- Realizar un diagrama de flujo que permita calcular la edad de una persona solicitando su fecha de nacimiento. Entrega impresa. Considerar limpieza y ortografía. Mínimo en una cuartilla.

5.- Realizar un diagrama de flujo que permita calcular el promedio de un alumno, el cual tiene 10 materias y cuya calificación se solicita previamente. Entrega impresa. Considerar limpieza y ortografía. Mínimo en una cuartilla.

6.- Realizar un diagrama de flujo que permita calcular la velocidad que emplea un móvil, considerando que velocidad=distancia/tiempo. Entrega impresa. Considerar limpieza y ortografía. Mínimo en una cuartilla.

7.- Realizar un diagrama de flujo que permita calcular la distancia que emplea un móvil, considerando que distancia=velocidad*tiempo. Entrega impresa. Considerar limpieza y ortografía. Mínimo en una cuartilla.

8.- Realizar un diagrama de flujo que permita calcular el tiempo que emplea un móvil, considerando que tiempo=distancia/velocidad. Entrega impresa. Considerar limpieza y ortografía. Mínimo en una cuartilla.

9.- Realizar un diagrama de flujo que permita calcular el área de un círculo. Entrega impresa. Considerar limpieza y ortografía. Mínimo en una cuartilla.

10.- Realizar avance de proyecto. Dependiendo del tipo de proyecto, el catedrático solicita los contenidos apropiados para este avance. La entrega será impresa. Considerar ortografía y limpieza.

11.- Realizar un diagrama de flujo que permita calcular el área de un cubo. Entrega impresa. Considerar limpieza y ortografía. Mínimo en una cuartilla.

12.- Realizar un diagrama de flujo que permita calcular la edad de una persona. Entrega impresa. Considerar limpieza y ortografía. Mínimo en una cuartilla.

13.- Realizar un diagrama de flujo que permita calcular el número de segundos de vida de una persona conociendo su fecha y hora de nacimiento. Entrega impresa. Considerar limpieza y ortografía.

14.- Realizar un diagrama de flujo que permita calcular el número total de kilómetros conociendo el número de milímetros. Considerar limpieza y ortografía. Entrega impresa.

15.- Realizar un diagrama de flujo que permita calcular el número de segundos transcurridos en cierto número de días solicitados. Entrega impresa. Considerar limpieza y ortografía.

16.- Realizar un diagrama de flujo que permita calcular el número de milímetros conociendo el número de pies solicitados previamente. Entrega impresa. Considerar limpieza y ortografía.

3.2. ESTRUCTURAS CONDICIONALES

Objetivo.

El alumno será capas de explicar la importancia de las estructuras condicionales en el diseño de algoritmos y resolverá problemas básicos mediante diagramas de flujo. Identificar su aplicación.

Estructuras condicionales[22]

En el diseño de algoritmos se presentan situaciones en las que una lista sencilla de instrucciones ya no resultan útiles para descripciones complicadas. Este es el caso cuando existen un número de posibles alternativas resultantes de la evaluación de una determinada condición. Las estructuras selectivas se utilizan para tomar decisiones lógicas: esta es la razón que se suelen denominar también *estructuras de decisión, condicionales o alternativas.*

En las estructuras condicionales se evalúa una condición y en función del resultado de la misma se realiza una opción u otra, señala Joyanes Aguilar. *Las condiciones se especifican usando expresiones lógicas*. La representación de una estructura selectiva se hace con palabras en pseudocódigo (*if, then*, *else* o bien en español *si, entonces, si_no*), con una figura geométrica en forma de rombo.

Las estructuras selectivas o alternativas pueden ser:

- *Simples,*
- *Dobles,*
- *Múltiples.*

[22] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 135.

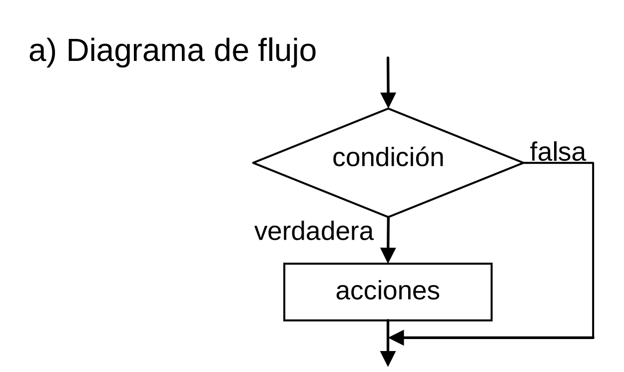
Alternativa Simple (Si-Entonces / If-Then)

La estructura alternativa simple *si-entonces* (en inglés *if-then*) ejecuta una determinada acción cuando se cumple una determinada condición. La selección *si-entonces* evalúa la condición y

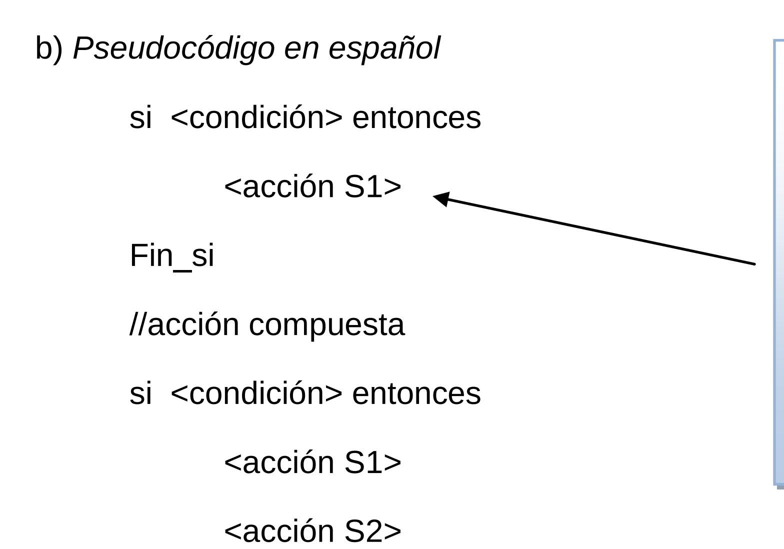
- Si la condición es *verdadera*, entonces ejecuta la acción S1 (o acciones caso de ser S1 una acción compuesta y constar de varias acciones).
- Si la condición es *falsa*, entonces no hacer nada.

Las representaciones gráficas de la estructura condicional simple se muestran en la figura 3.2.1.

a) Diagrama de flujo



b) *Pseudocódigo en español*



```
si  <condición> entonces
        <acción S1>
Fin_si
//acción compuesta
si  <condición> entonces
        <acción S1>
        <acción S2>
Fin_si
```

<u>Sugerencia:</u> Se recomienda dejar esta identación o sangrado al redactar los algoritmos, para facilitar la lectura. Así mismo, se sugiere aplicar esta estrategia al capturar los programas, sobre todo cuando existe mucho código.

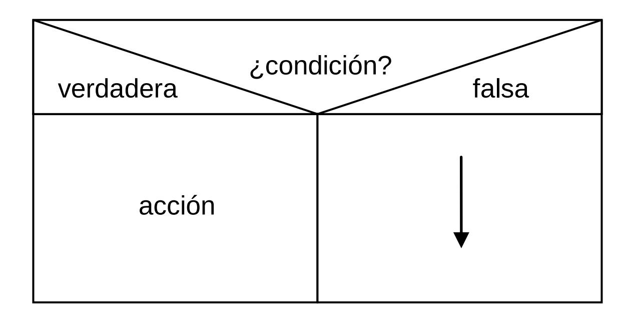
Pseudocódigo en inglés

If <condición> then

<acción S1>

Endif

c) *Diagrama N-S*



d) Sintaxis en C/C++

If (condición)

{ //sentencias }

Figura 3.2.1. Estructura alternativas simples.

Ejemplo 1: Algoritmo para identificar el mayor de dos números.

Representación del algoritmo en Pseudocódigo

Inicio

Leer(numero1)

Leer(numero2)

Si numero1>numero2 entonces

Escribir("El mayor es número 1")

Fin_si

Si numero2>numero1 entonces

Escribir("El mayor es número 2")

Fin_si

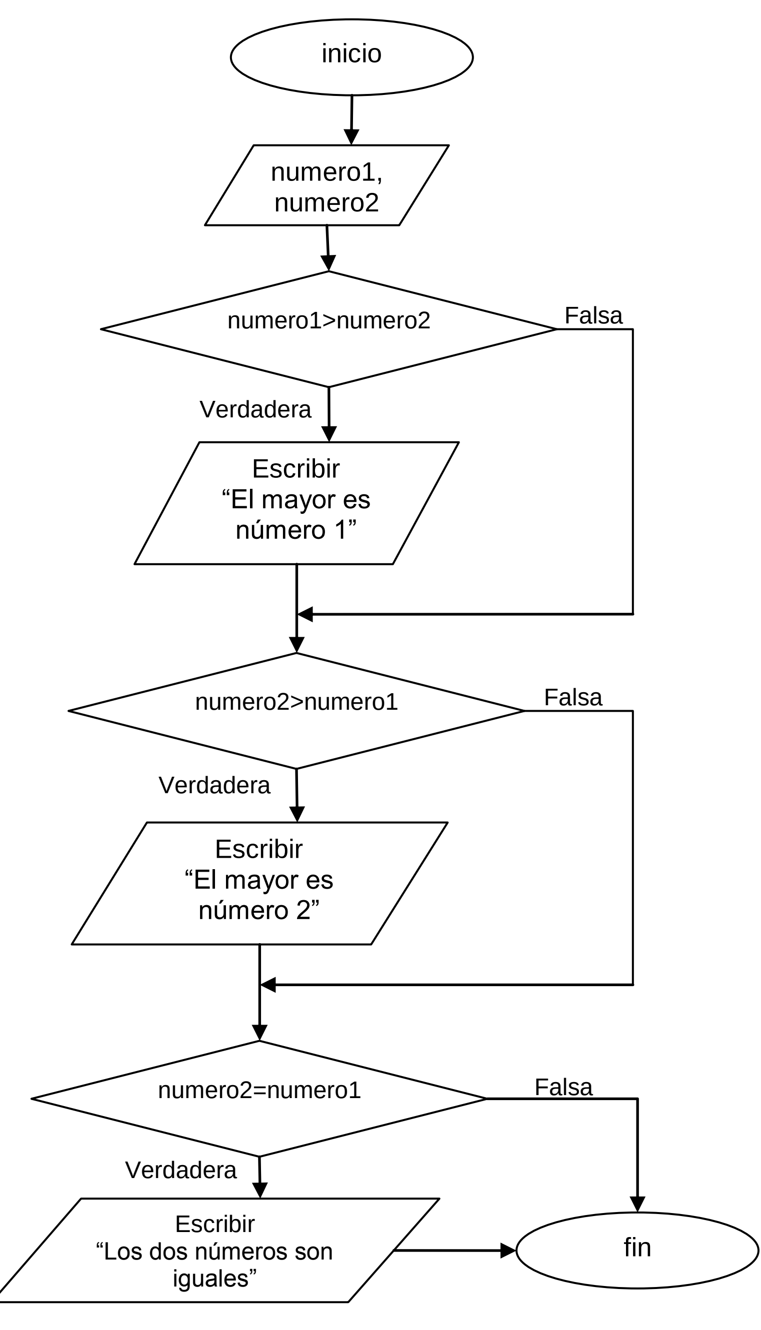
Si numero1=numero2 entonces

Escribir(“Los dos números son iguales”)

Fin_si

Fin

Diagrama de flujo



Código en C++ del ejemplo anterior:

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main(){
        int numero1,numero2; //declaración de variables de tipo entero
        clrscr(); // función que limpia la pantalla
        cout<<"Introduce primer numero:";  //imprime texto en pantalla
        cin>>numero1;    //lee un dato desde teclado
        cout<<"Introduce segundo numero:";  //imprime texto en pantalla
        cin>>numero2;    //lee un dato desde teclado
        if(numero1>numero2){cout<<"El mayor es numero 1";}
        if(numero2>numero1){cout<<"El mayor es numero 2";}
        if(numero1==numero2){cout<<"Los dos números son iguales";}
        getch ();
        }
```

Ejemplo 2: Algoritmo para calcular el total a pagar a los empleados por día, considerando que la jornada normal comprende 8 horas con un costo de 100 por hora, si se laboran horas extras el costo por hora es de 150.

Pseudocódigo

```
Inicio

        total←0

        Leer(horas)

        Si (horas>8) entonces

        Extra←horas-8

        Total←800+(extra*150)

        Fin_si

        Si (horas>0 y horas<=8) entonces

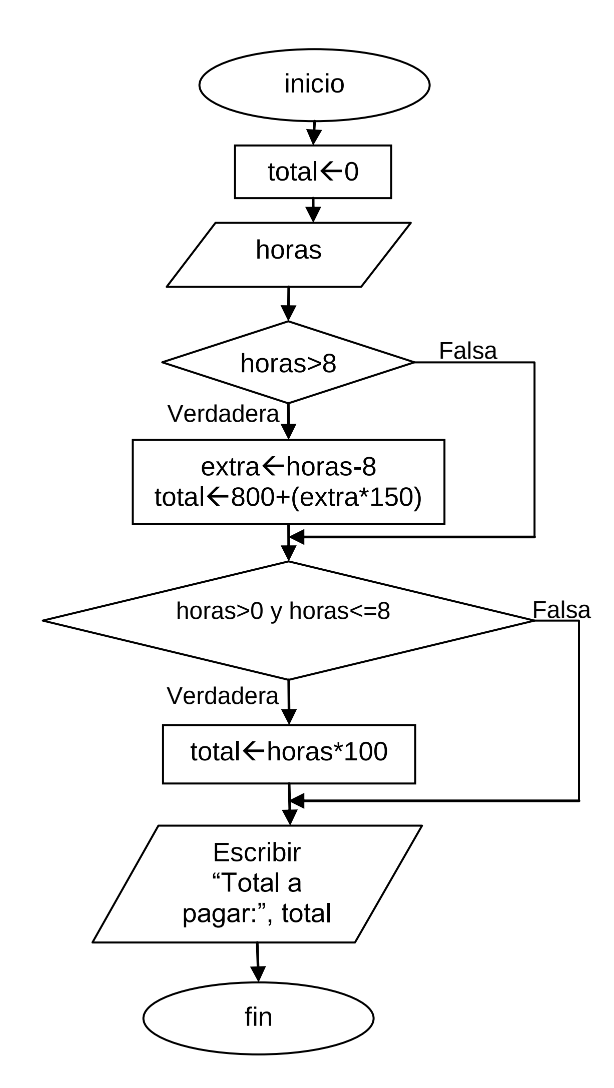
        total←horas*100

        Fin_si

        Escribir ("Total a pagar:", total)

        Fin
```

Diagrama de flujo



Código en C++ del ejemplo anterior:

```
#include<stdio.h>
```

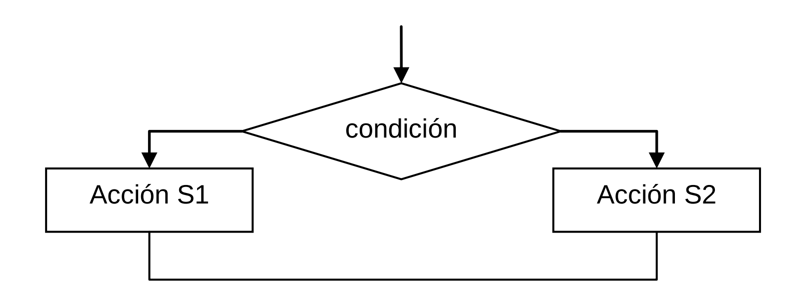
```
#include<conio.h>
#include<iostream.h>
void main(){
        float total, horas, extra; //declaración de variables de tipo real
        clrscr(); // función que limpia la pantalla
        total = 0;
        cout<<"Horas laboradas:";  //imprime texto en pantalla
        cin>>horas;    //lee un dato desde teclado
        if(horas>8){extra = horas-8;total = 800+(extra*150);}
        if(horas>0 && horas<=8){total=horas*100;}
        cout<<"Total a pagar:"<<total;
        getch ();}
```

Alternativa doble (si-entonces-sino / if-then-else)[23]

La estructura anterior es muy limitada y normalmente se necesitará una estructura que permita elegir entre dos opciones o alternativas posibles, en función del cumplimiento o no de una determinada condición. Si la condición *c* es verdadera, se ejecuta la acción S1 y, si es falsa, se ejecuta la acción S2. Las representaciones gráficas de la estructura condicional doble se muestran en la figura 2.

Figura 2. Representaciones de estructura condicional doble.

a) Diagrama de flujo



b) *Pseudocódigo en español*

Si <condición> entonces

[23] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 136.

```
    <acción S1>

Si_no

    <acción S2>

Fin_si

//acción compuesta

Si <condición> entonces

    <acción S11>

    <acción S12>

    .
    .
    .
    <acción S2n>

Si_no

    <acción S21>

    <acción S22>

    .
    .
    .
    <acción S1n>
```

Fin_si

Pseudocódigo en inglés

If <condicion> then

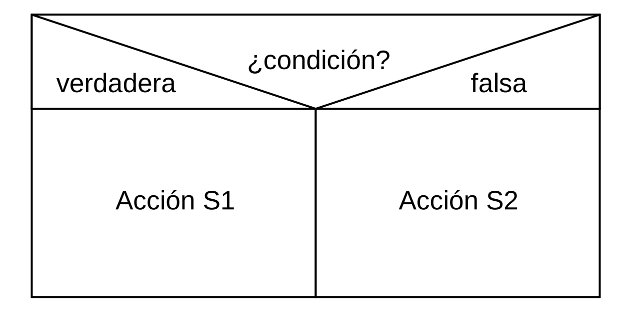
<acción S1>

Else

<acción S2>

Endif

c) *Diagrama N-S*



d) Sintaxis en C/C++

If (condición)

{//sentencia(s)}

Else

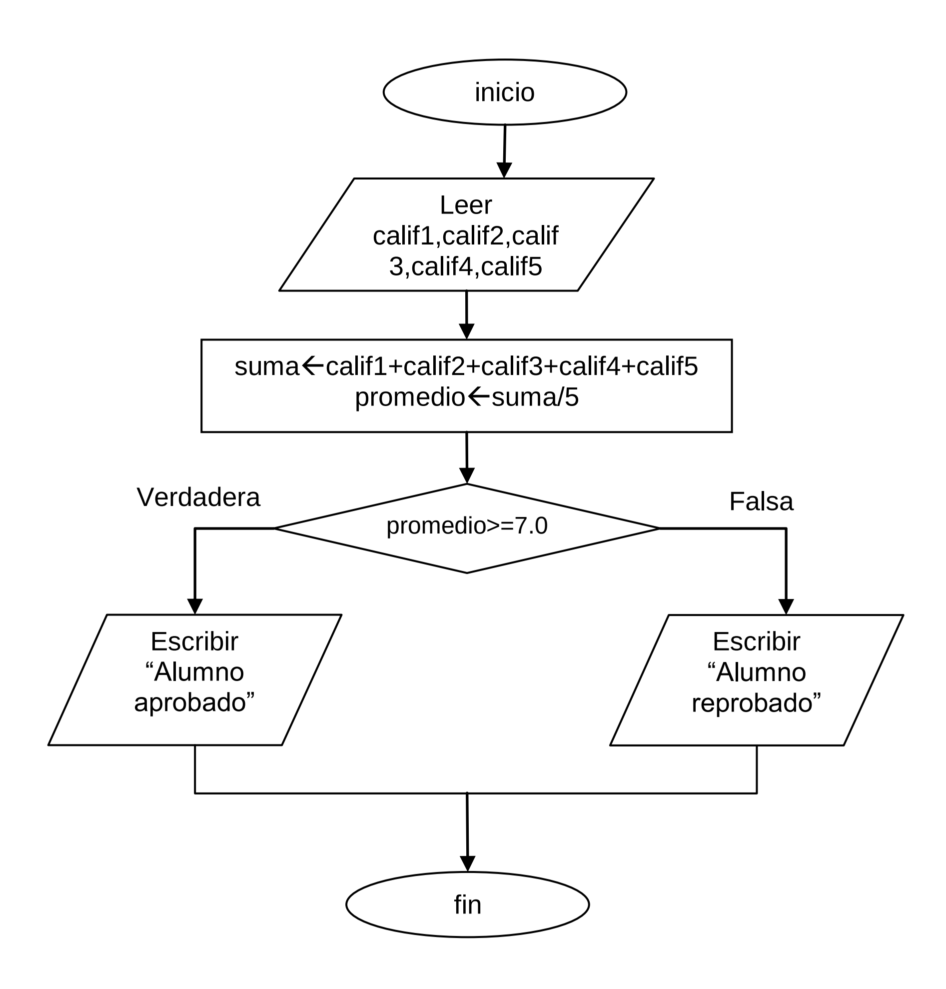
{//sentencia(s)}

Ejemplo 1: Algoritmo para decidir si un alumno está aprobado considerando que cursa cinco asignaturas y el promedio de aprobación es de 7.0

Pseudocódigo

```
Inicio
    Leer (calif1,calif2,calif3,calif4,calif5)
    suma←calif1+calif2+calif3+calif4+calif5
    promedio←suma/5
    Si promedio>=7.0 entonces
        Escribir (“Alumno aprobado”)
    Si_no
        Escribir (“Alumno reprobado”)
    Fin_si
    fin
```

Diagrama de flujo



Código en C++ del ejemplo anterior:

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main(){
        float calif1,calif2,calif3,calif4,calif5,suma,promedio;
        clrscr(); //función que limpia la pantalla
        cout<<”Introduce calificación 1:”; cin>>calif1;
```

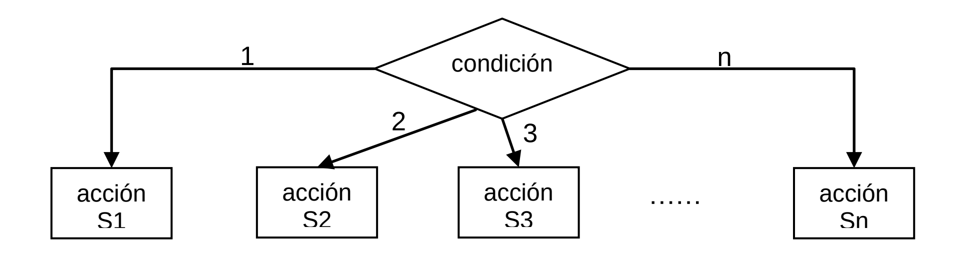
```
cout<<"Introduce calificación 2:"; cin>>calif2;
cout<<"Introduce calificación 3:"; cin>>calif3;
cout<<"Introduce calificación 4:"; cin>>calif4;
cout<<"Introduce calificación 5:"; cin>>calif5;
suma=(calif1+calif2+calif3+calif4+calif5);
promedio=suma/5;
if(promedio>=7.0)
{cout<<"Alumno aprobado";}
else
{cout<<"Alumno reprobado";}
getch ();}
```

Alternativa múltiple (según sea, caso de / case)[24]

Con frecuencia en la práctica, es necesario que existan más de dos elecciones posibles.

La estructura de decisión múltiple evalúa una expresión que puede tomar *n* valores distintos, 1, 2, 3, 4,…, n. Según se elija uno de estos valores en la condición, se realizará una de las *n* acciones, o lo que es igual, el flujo del algoritmo seguirá un determinado camino entre los *n* posibles. Las representaciones de la estructura condicional múltiple se representan en la figura 3.2.3.

a) *Diagrama de flujo*



b) *Pseudocódigo*

Según_sea *expresión (E)* hacer

---

[24] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 142.

E1: acción S11

acción S12

acción S1a

E2: acción S21

acción S22

.

.

acción S2b

En: acción S31

acción S32

.

.

acción S3p

si-no

acción Sx

fin_segun

c) *Sintaxis* (C, C++, Java, C#)

```
swtich (expresión)

{case valor1:

        Sentencia1;

        Sentencia2;

        .
        .
        break;

    case valor2:

        Sentencia1;

        Sentencia2;

        .
        .
        break;

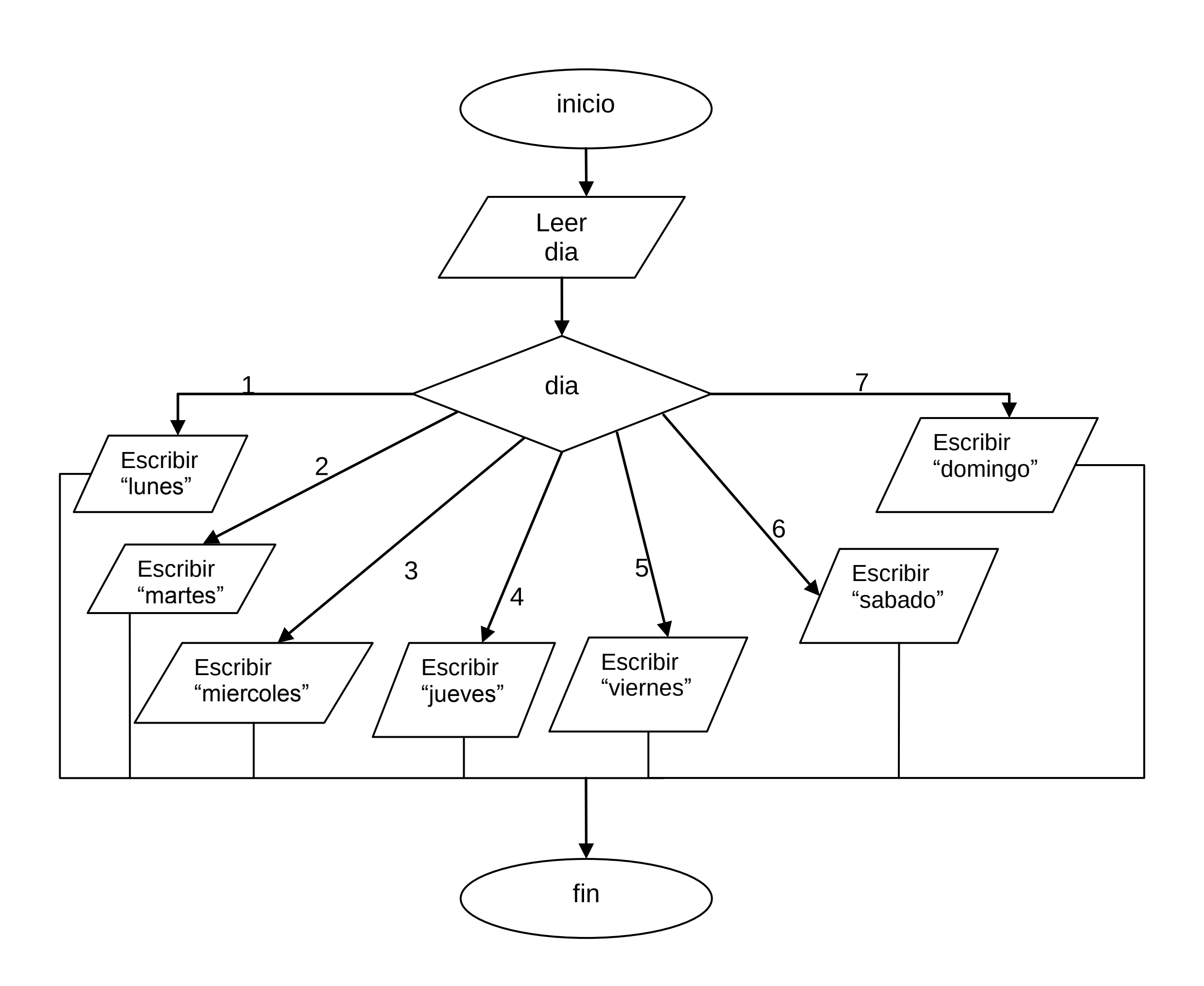
    default:
        Sentencia1;
        Sentencia2;
        .
        .}
```

Ejemplo 1: Se desea diseñar un algoritmo que escriba los nombres de los días de la semana en función del valor de una variable DIA introducida por teclado.

Pseudocódigo

```
Inicio
    Leer (día)
    Según_sea día hacer
        1: escribir (“lunes”)
        2: escribir (“martes”)
        3: escribir (“miércoles”)
        4: escribir (“jueves”)
        5: escribir (“viernes”)
        6: escribir (“sábado”)
        7: escribir (“domingo”)
        Si_no
            escribir (“Opción no disponible”)
    fin_segun
fin
```

Diagrama de flujo.



Código en C++ del ejemplo anterior:

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main(){
        int día;
        clrscr();
        cout<<”Pulsa un número del (1-7) que corresponde a un día\n”;
        cin>>día;
```

```
        switch(día)
        {
                case 1: cout<<"lunes"; break;
                case 2: cout<<"martes"; break;
                case 3: cout<<"miércoles"; break;
                case 4: cout<<"jueves"; break;
                case 5: cout<<"viernes"; break;
                case 6: cout<<"sábado"; break;
                case 7: cout<<"domingo"; break;
                default: cout<<"opción no disponible"; //pulsar número equivocado
        }

        getch ();
}
```

Estructuras de decisión anidadas (en escalera)[25]

Las estructuras de decisión si-entonces y si-entonces-s_no implican la selección de una de dos alternativas. Es posible también utilizar la instrucción **si** para diseñar estructuras de selección que contengan más de dos alternativas. Por ejemplo, una estructura **si-entonces** puede contener otra, y así sucesivamente cualquier número de veces; a su vez, dentro de cada estructura pueden existir diferentes acciones.

a) *Pseudocódigo (anidadas o encajadas)*

```
Si <condicion1> entonces
        Si <condicion2> entonces
        .
        .
        .
        Fin_si
Fin_si
```

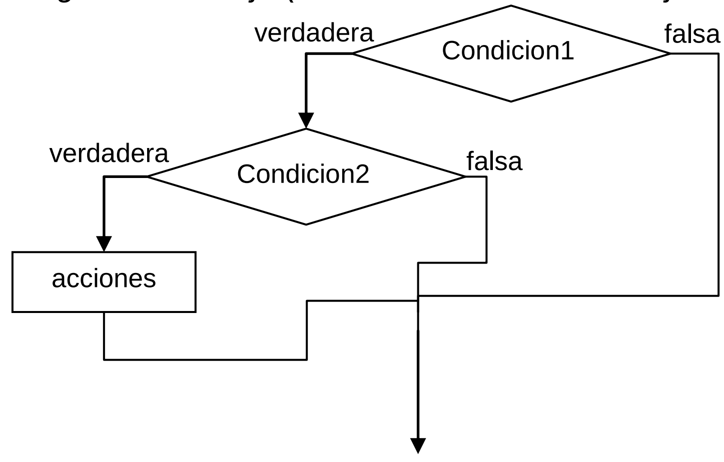
Pseudocódigo (n alternativas o de decisión múltiple)

```
Si <condicion1> entonces
        <acciones>
Sin_no
        Si condicion2 entonces
```

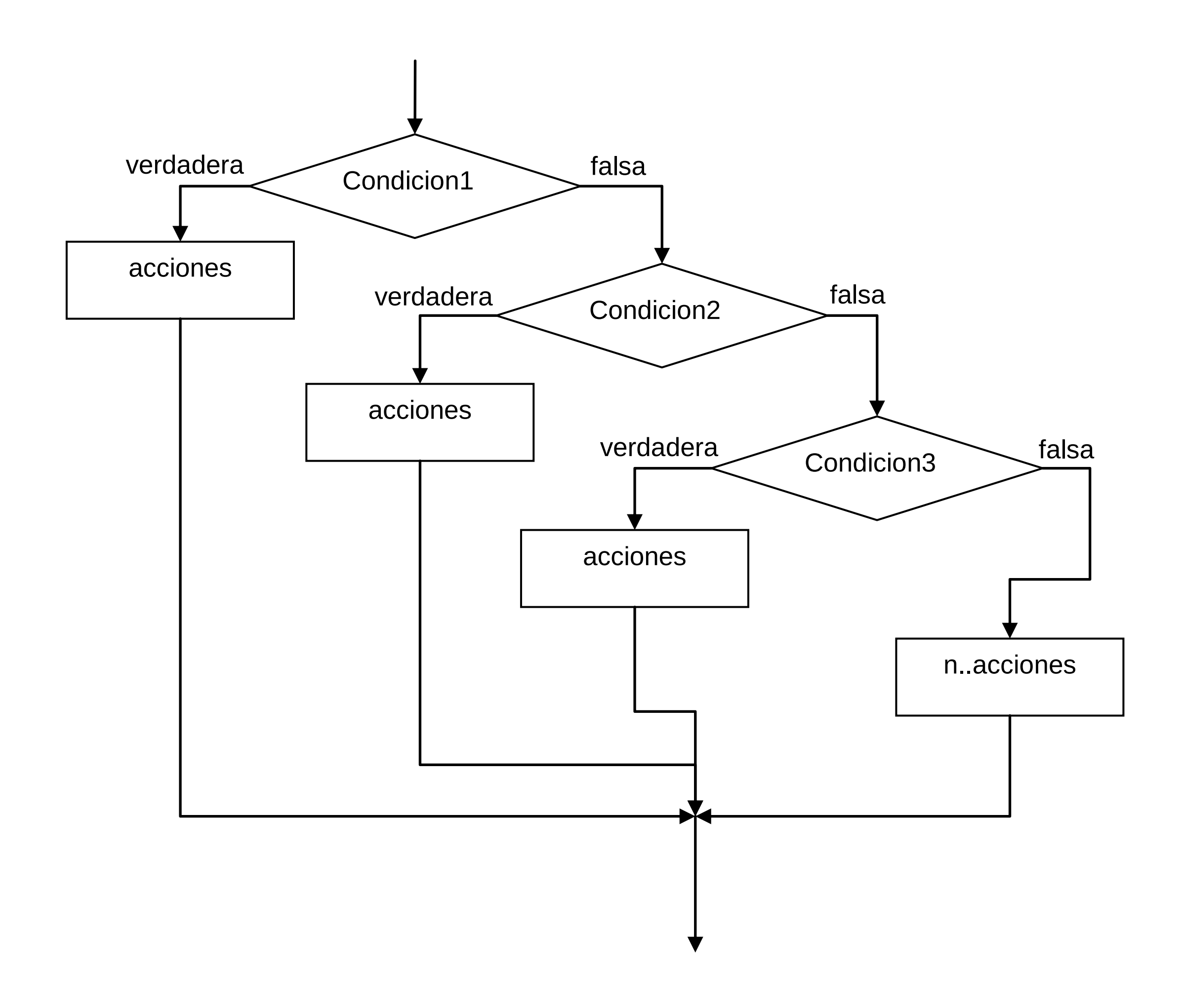
[25] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 149.

```
        <acciones>
    Si_no
        Si condicion3 entonces
            <acciones>
        Si_no
        .
        .
        .
        Fin_si
    Fin_si
Fin_si
```

b) *Diagramas de flujo (estructura anidada o encajada)*



Diagramas de flujo (n alternativas o de decisión múltiple)



c) Sintaxis en C++ *(estructuras de decisión anidadas o encajadas)*

```
if (condicion1)
{If(condicion2)
        {

        }
}
```

Código en C++ *(n alternativas o de decisión múltiple)*

```
if(condicion1)
```

```
{acciones}
else if (condicion2)
{acciones}
else if(condicion3)
{acciones}
else
{acciones}
```

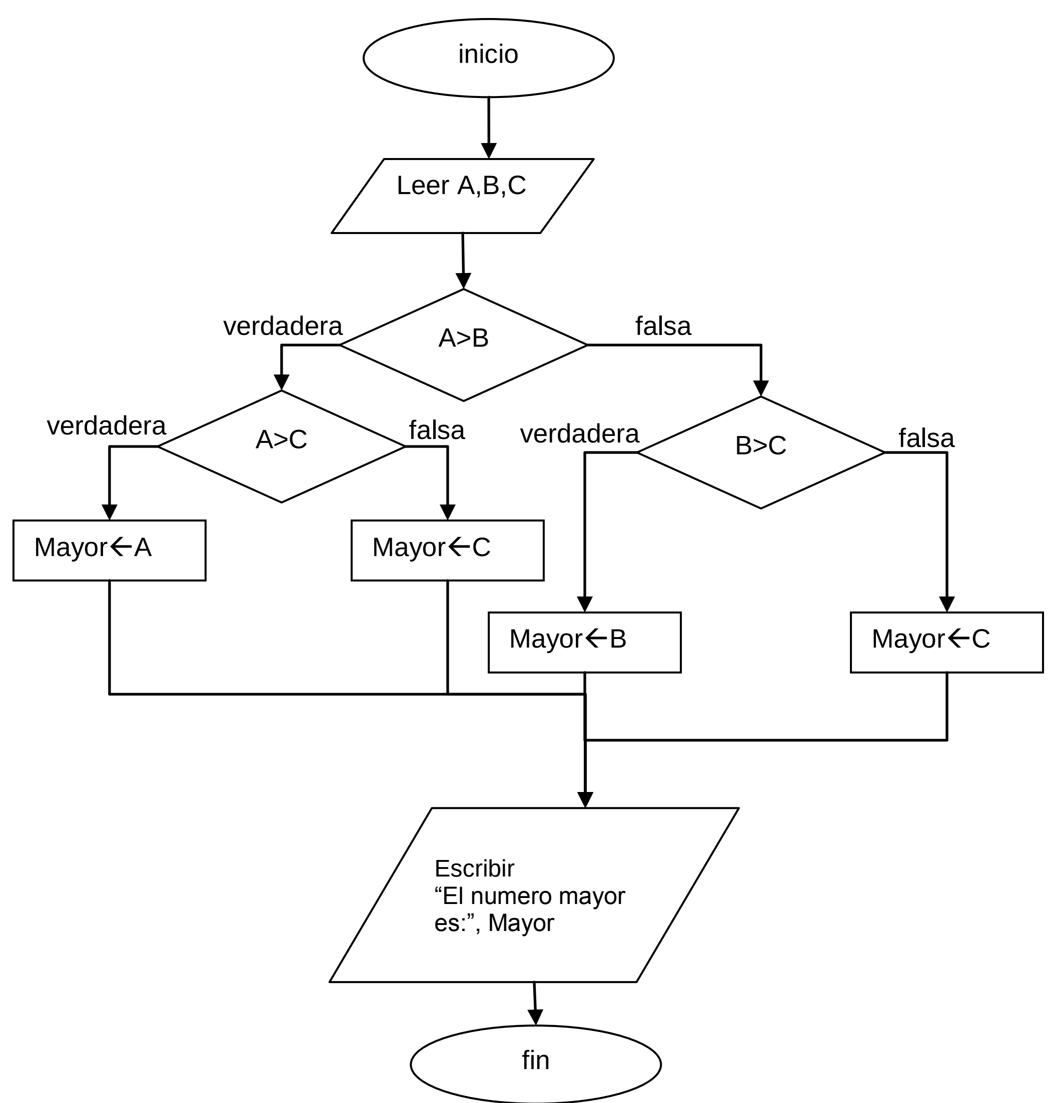
Ejemplo 1: Diseñar un algoritmo que lea tres números A, B, C y visualice en pantalla el valor más grande. Se supone que los tres valores son diferentes.

Los tres números son A, B y C; para calcular el más grande se realizarán comparaciones sucesivas por parejas.

Pseudocódigo

```
Inicio
    Leer(A,B,C)
    Si A>B entonces
        Si A>C entonces
            Mayor←A
        Si_no
            Mayor←C
        Fin_si
    Si_no
        Si B>C entonces
            Mayor←B
        Si_no
            Mayor←C
        Fin_si
    Fin_si
    Escribir("El numero mayor es:", Mayor)
Fin
```

Diagrama de flujo



Código en C++ del ejemplo anterior.

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
```

```
void main(){
        int A,B,C,Mayor;
        clrscr();
        cout<<"\nIntroduce primer número llamar A: "; cin>>A;
        cout<<"\nIntroduce segundo numero llamar B: "; cin>>B;
        cout<<"\nIntroduce tercer numero llamar C: "; cin>>C;
        if(A>B){
                if(A>C)
                {Mayor=A;}
                else
                {Mayor=C;}
                }
                else
                {if(B>C){
                Mayor=B;
                }
                else
                {Mayor=C;}
                }
                cout<<"\nEl numero mayor es:"<<Mayor;
                getch();
                }
```

Ejemplo 2: Realizar un algoritmo que permita solicitar 3 calificaciones, calcular el promedio. Si el promedio se ubica en los siguientes rangos, escribir los correspondientes mensajes. Considerar la situación en donde el promedio generado no esté contemplado en los rangos establecidos.

9.5-10.0 Excelente

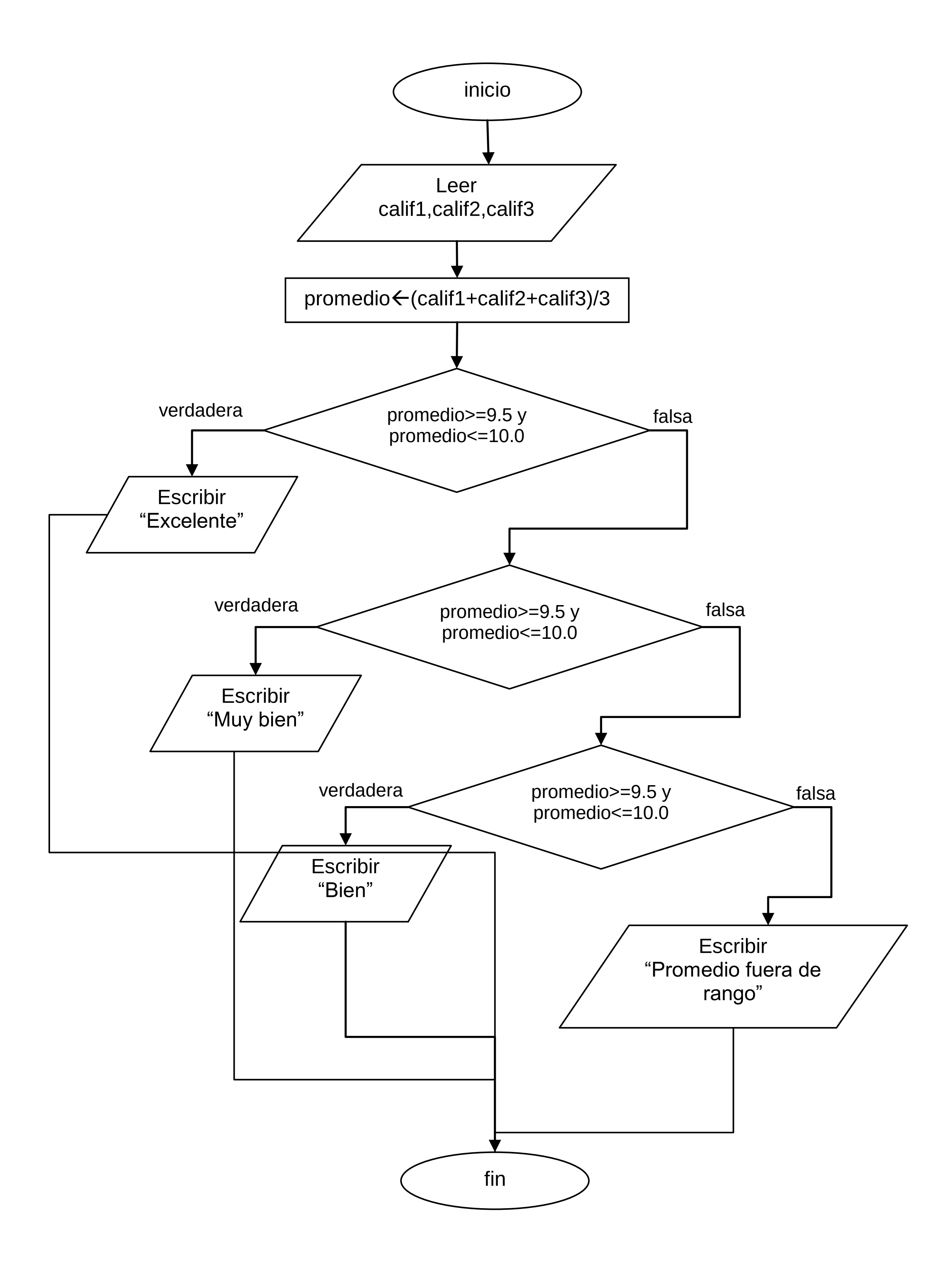
8.5-9.4 Muy bien

7.5-8.4 Bien

Pseudocódigo

```
Inicio
        Leer(calif1, calif2,calif3)
        promedio←(calif1+calif2+calif3)/3
        Si (promedio>=9.5 y promedio<=10.0) entonces
                Escribir("Excelente")
        Si_no
```

```
        Si (promedio>=8.5 y promedio<=9.4) entonces
            Escribir("Muy bien")
        Si_no
            si(promedio>=7.5 y promedio<=8.4) entonces
                Escribir("Bien")
            Si_no
                Escribir("Promedio fuera de rango")
            Fin_si
        Fin_si
    Fin_si
Fin
```


inicio
Leer
calif1,calif2,calif3
promedio←(calif1+calif2+calif3)/3
verdadera
promedio>=9.5 y
promedio<=10.0
falsa
Escribir
“Excelente”
verdadera
promedio>=9.5 y
promedio<=10.0
falsa
Escribir
“Muy bien”
verdadera
promedio>=9.5 y
promedio<=10.0
falsa
Escribir
“Bien”
Escribir
“Promedio fuera de
rango”
fin

Código en C++ del ejemplo anterior.

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main(){
	float calif1,calif2,calif3,promedio;
	clrscr();
	cout<<"\nIntroduce primera calificación:"; cin>>calif1;
	cout<<"\nIntroduce segunda calificación:"; cin>>calif2;
	cout<<"\nIntroduce tercera calificación:"; cin>>calif3;
	promedio=(calif1+calif2+calif3)/3;
	if(promedio>=9.5 && promedio<=10.0)
	{
		cout<<"\nExcelente");
	}
	else if(promedio>=8.5 && promedio<=9.4)
	{
		cout<<"\nMuy bien");
```

```
}
else if(promedio>=7.5 && promedio<=8.4)
{
        cout<<”\nBien”);

}
else
{
        cout<<”\nPromedio fuera de rango”;
}
getch();
}
```

ACTIVIDADES DE APRENDIZAJE

1.- Realizar un diagrama de flujo que solicite la edad de una persona, en base a este dato imprimir si la persona es "mayor de edad"; considerando que la mayoría de edad es a partir de los 18 años. Entrega impresa. Considerar ortografía y limpieza.

2.- Realizar un diagrama de flujo que solicite un número, en base a este dato imprimir si el número "es positivo". Entrega impresa. Considerar ortografía y limpieza.

3.- Realizar un diagrama de flujo que solicite el nombre de un alumno y cinco calificaciones; con estos datos calcular el promedio del alumno. Imprimir si el alumno está aprobado considerando que el promedio aprobatorio es mayor o igual a 7.0. Entrega impresa. Considerar ortografía y limpieza.

4.- Realizar un diagrama de flujo que solicite el nombre de un alumno y cinco calificaciones; con estos datos calcular el promedio del alumno. Si el promedio se ubica en el rango de 9.5-10.0 imprimir "excelente", en caso contrario si el promedio se ubica en el rango de 8.5-9.4 imprimir "muy bien”, en caso contrario si el promedio se ubica en el rango de 7.5-8.4 imprimir "bien”, en caso contrario si el promedio se ubica en el rango de 7.0-7.4 imprimir "regular”. Entrega impresa. Considerar ortografía y limpieza.

5.- Realizar un diagrama de flujo que imprima las siguientes opciones "1.- suma 2.- resta 3.- multiplicación 4.- división". Posteriormente debe solicitar dos números, así mismo debe solicitar el número de la operación a realizar sobre los números solicitados previamente. Es decir, si la opción seleccionada es 1 debe realizar la suma de los números e imprimir el resultado. Si la opción seleccionada es 2 debe realizar la resta de los números e imprimir el resultado. Si la opción seleccionada es 3 debe realizar la multiplicación de los números e imprimir el resultado. Si la opción seleccionada es 4 debe realizar la división de los números e imprimir el resultado. La entrega de la actividad será impresa. Considerar la ortografía y limpieza.

6.- Realizar el programa del diagrama de flujo anterior. Entrega impresa del código del programa generado, así como imprimir la pantalla de salida que genera el programa una vez ejecutado.

3.3. CONTADORES

Objetivo

El estudiante explicará la importancia de los contadores en los algoritmos. Identificará la aplicación de los contadores Y será capaz de resolver problemas básicos mediante diagramas de flujo.

Contadores[26]

Se define como contador *aquella variable que es utilizada en un ciclo repetitivo* y tiene por objetivo almacenar valores cuyos incrementos o decrementos son en forma *constante* por cada iteración de ciclo o *bucle* en cuestión.

Por lo general los contadores se emplean en los ciclos para controlar el número de iteraciones en los mismos, o para almacenar, totales de elementos.

Ejemplo de contadores:

Con incremento (por ejemplo, incremento constante de una unidad)

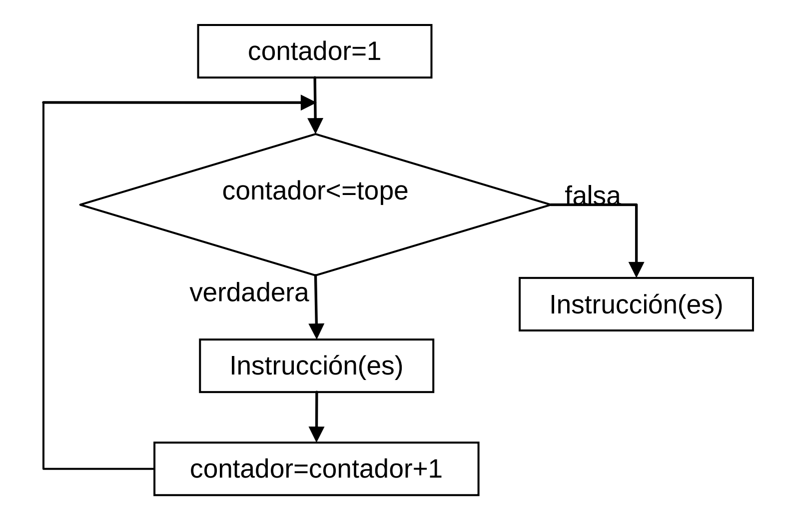
contador=contador+1

Con decremento (por ejemplo, decremento constante de una unidad)

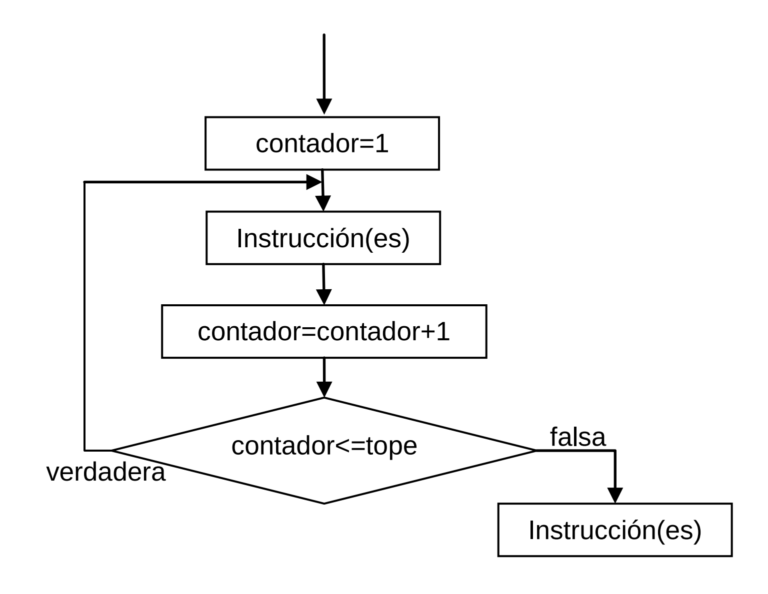
contador=contador-1

[26] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 168.

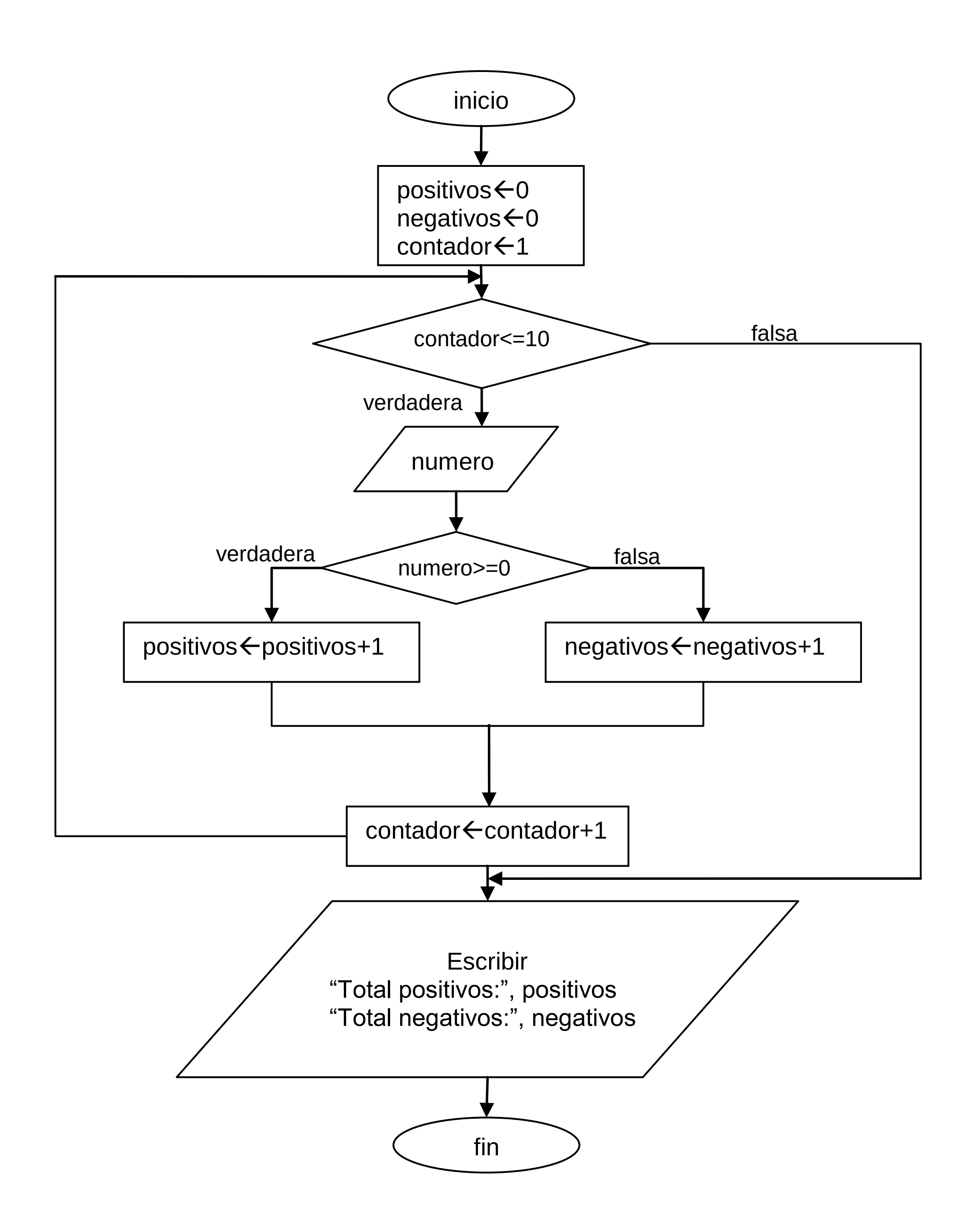
Estructura repetitiva ***(while)*** que emplea contador



Estructura repetitiva ***(do-while)*** que emplea contador



Ejemplo: Algoritmo que contabiliza la cantidad de números positivos y negativos a partir de 10 números introducidos por teclado. Se recomienda al alumno leer el tema de ciclos para comprender más ampliamente el tema.


inicio
positivos←0
negativos←0
contador←1
contador<=10
falsa
verdadera
numero
verdadera
numero>=0
falsa
positivos←positivos+1
negativos←negativos+1
contador←contador+1
Escribir
“Total positivos:”, positivos
“Total negativos:”, negativos
fin

ACTIVIDADES DE APRENDIZAJE

1.- Realizar un diagrama de flujo que solicite cinco números. Posteriormente debe imprimir cuantos números de los introducidos fueron positivos y cuantos números fueron negativos. La entrega del diagrama de flujo será impresa. Considerar la ortografía y limpieza.

2.- Realizar un diagrama de flujo que solicite diez números. Posteriormente debe imprimir cuantos números ubicados en el rango de 1-10 fueron introducidos, cuantos números del 11-100 fueron introducidos y cuántos números mayores a 100 fueron introducidos. La entrega del diagrama de flujo será impresa. Considerar la ortografía y limpieza.

1.- Realizar un diagrama de flujo que solicite 10 números. Posteriormente debe imprimir cuantos números de los introducidos fueron pares y cuantos números fueron impares. La entrega del diagrama de flujo será impresa. Considerar la ortografía y limpieza.

2.- Realizar un diagrama de flujo que solicite diez números. Posteriormente debe imprimir cuantos números ubicados en el rango de 10-20 fueron introducidos, cuantos números del 21-500 fueron introducidos y cuántos números mayores a 501 fueron introducidos. La entrega del diagrama de flujo será impresa. Considerar la ortografía y limpieza.

3.4. ACUMULADORES

Objetivo

El estudiante podrá explicar la importancia de los acumuladores en los algoritmos, así mismo identificará las aplicaciones de los acumuladores y resolverá problemas básicos mediante diagramas de flujo.

Acumuladores[27]

Se define como acumulador *aquella variable que es utilizada en un ciclo repetitivo* y tiene por objetivo almacenar valores cuyos incrementos o decrementos son en forma *variable* por cada iteración de ciclo o *bucle* en cuestión.

Por lo general los contadores se emplean en los ciclos para controlar el número de iteraciones en los mismos, o para almacenar totales de elementos.

Ejemplo de acumuladores:

Con incremento (por ejemplo, incremento variable representado por *x*)

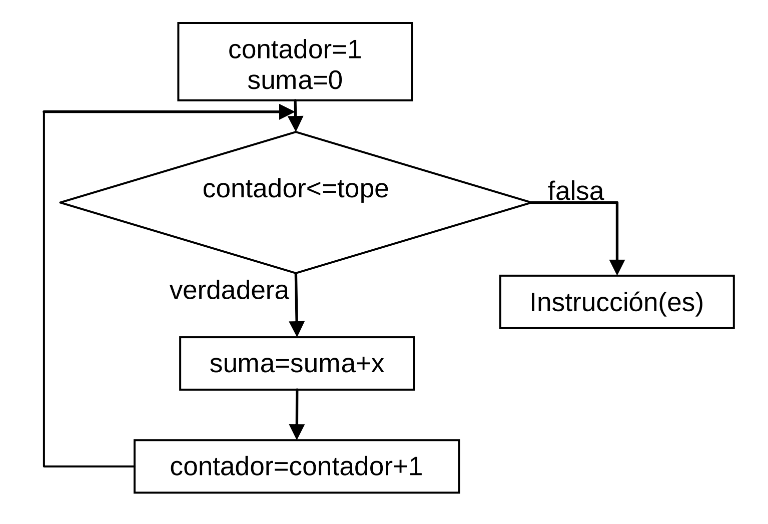
contador=contador+x

Con decremento (por ejemplo, decremento variable representado por *x*)

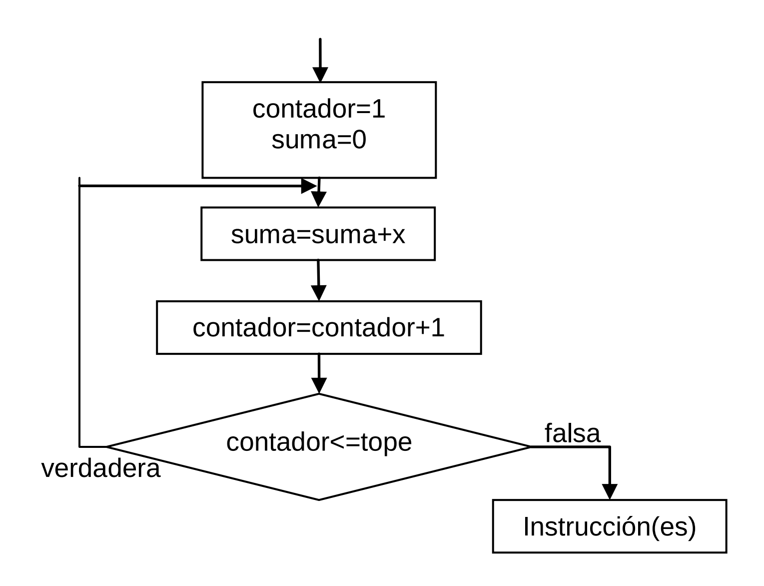
contador=contador-y

[27] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 167.

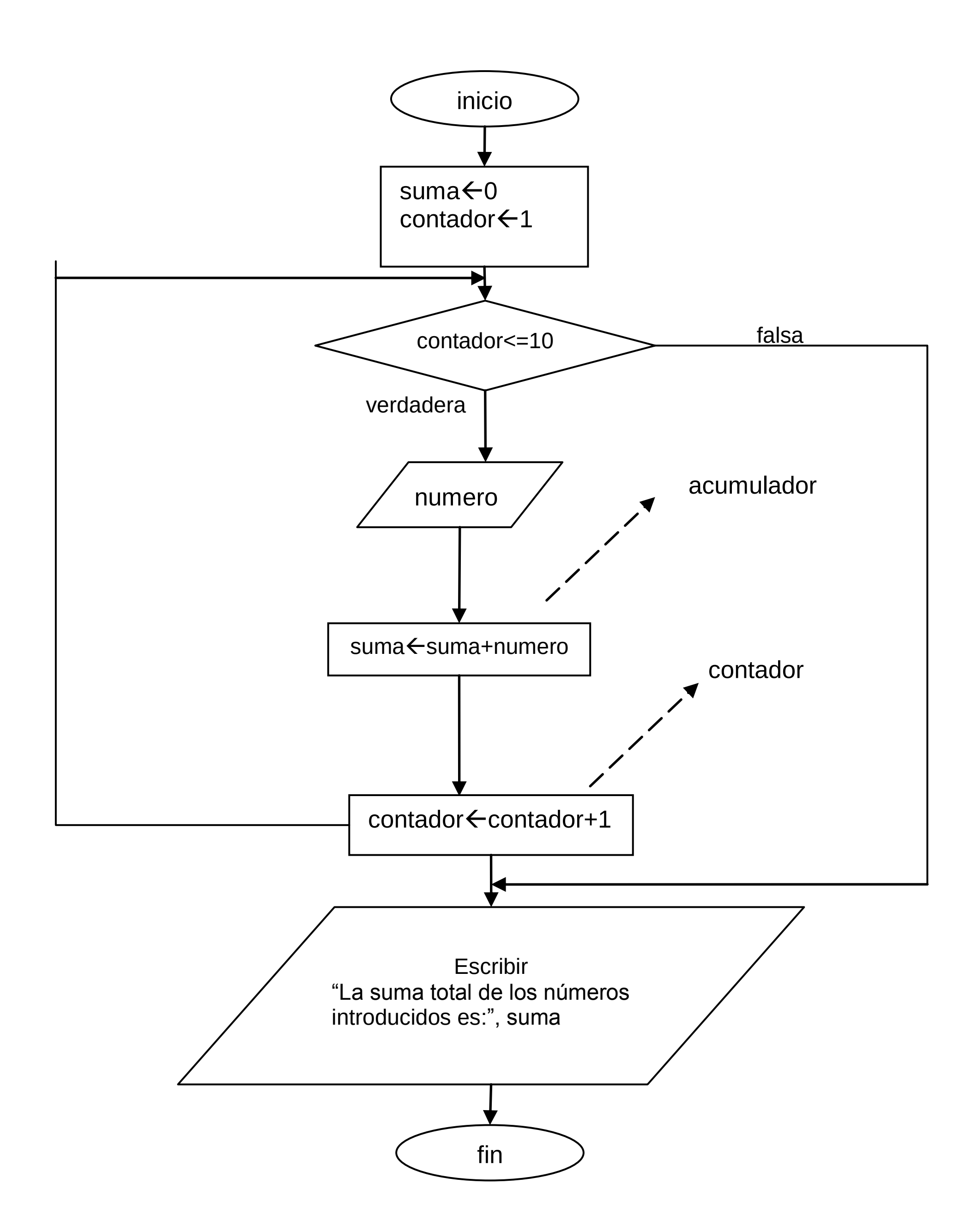
Estructura repetitiva ***(while)*** que emplea contador



Estructura repetitiva ***(do-while)*** que emplea contador



Ejemplo: Algoritmo que calcula la suma de un total de 10 números introducidos por teclado. Se recomienda al alumno leer el tema de ciclos para comprender más ampliamente el tema.


inicio
suma←0
contador←1
contador<=10
falsa
verdadera
numero
acumulador
suma←suma+numero
contador
contador←contador+1
Escribir
“La suma total de los números
introducidos es:”, suma
fin

ACTIVIDADES DE APRENDIZAJE

1.- Realizar un diagrama de flujo que solicite el nombre de un alumno, y solicite la calificación de cinco materias. Así mismo, solicitar el nombre de otro alumno y solicitar sus correspondientes calificaciones de sus cinco materias también. Al final se debe imprimir el promedio de cada alumno y el promedio general de los dos. La entrega del diagrama será impresa. Considerar ortografía y limpieza.

2.- Realizar avance de proyecto. Dependiendo del tipo de proyecto, el catedrático solicita los contenidos apropiados para este avance. La entrega será impresa. Considerar ortografía y limpieza.

3.5. CICLOS.

Objetivo.

Los participantes podrán exponer las características e importancia de los ciclos en los algoritmos, identificarán su aplicación y serán capaces de resolver problemas básicos mediante diagramas de flujo.

Ciclos[28]

Las computadoras están diseñadas especialmente para todas aquellas aplicaciones en las cuales una operación o conjunto de ellas deben repetirse muchas veces, *Joyanes Aguilar*. Señala además que un tipo importante de estructura es el algoritmo necesario para repetir una o varias acciones un número determinado de veces.

Las estructuras que repiten una secuencia de instrucciones, un número determinado de veces, se nombra *bucles,* y se denomina *iteración* al hecho de repetir la ejecución de una secuencia de acciones.

Dentro de los ciclos repetitivos más comunes se encuentra el ciclo while, do, while y for.

Existen acciones a realizar en donde se presta para aplicar los ciclos, por ejemplo, si se desea imprimir todos los números primos del 1 al 1000, en vez de escribir, los números 1 a 1, es posible diseñar un algoritmo que tenga un bucle del 1 al 1000 y dentro del mismo se aplique la lógica correspondiente para identificar los números primos.

Así mismo si se desea sumar todos los números comprendidos del 10-10000 resultaría una actividad laboriosa llevar a cabo esa cuenta elemento por

[28] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 163.

elemento por lo que amerita la aplicación de algún ciclo repetitivo que ahorre los pasos individuales.

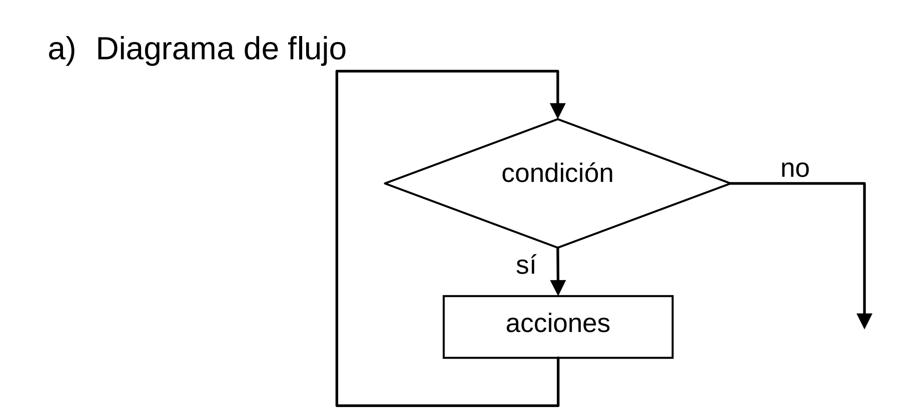
3.5.1 WHILE[29]

Joyanes Aguilar señala que la estructura repetitiva mientras (en inglés while) es aquella en la que el cuerpo del bucle se repite mientras se cumple una determinada condición. Cuando se ejecuta la instrucción **mientras**, la primera cosa que sucede es que se evalúa la condición (*una expresión booleana*). Si se evalúa falsa, no se toma ninguna acción y el programa prosigue en la siguiente instrucción del bucle. Si la expresión *booleana* es *verdadera*, entonces se ejecuta el cuerpo del bucle, después de lo cual se evalúa de nuevo la expresión booleana. Este proceso se repite una y otra vez **mientras** la expresión *booleana* (condición) sea verdadera.

La característica fundamental del ciclo while es que para ejecutarse por lo menos una vez el cuerpo de instrucciones se debe cumplir la condición, es decir, la *expresión booleana.*

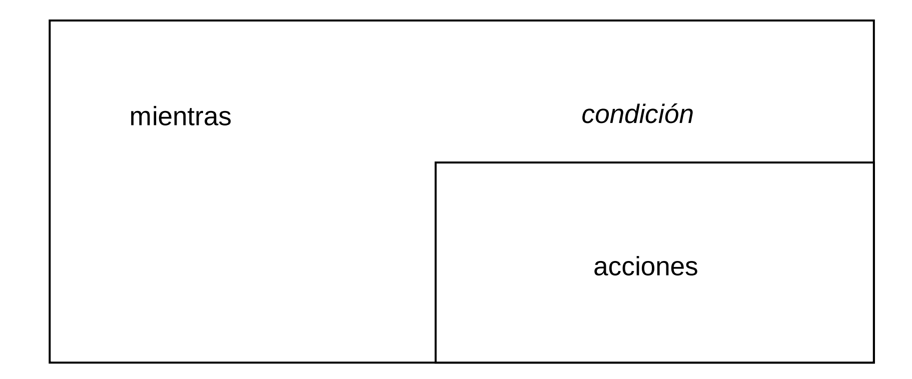
Las representaciones gráficas son:

a) Diagrama de flujo



[29] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 166.

b) Diagrama N-S



c) Pseudocódigo

```
mientras condición hacer
    acción S1
    acción S2
    .
    .
    acción Sn
fin_mientras
```

d) Sintaxis en C++

```
while (condición)
{

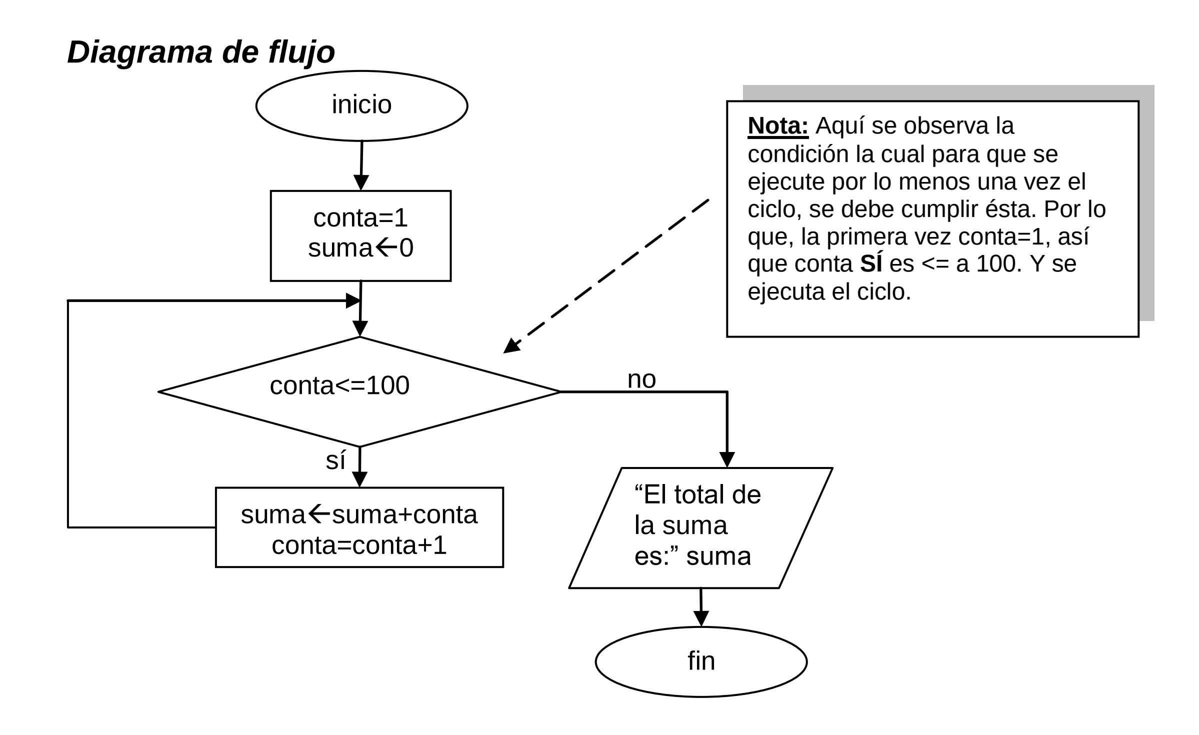
    instrucción(es);
}
```

Ejemplo 1: Algoritmo que permita obtener la suma de todos los números contenidos del 1-100.

Pseudocódigo

```
Inicio
        suma←0
        conta←1
        mientras conta<=100 hacer
                suma←suma+conta
                conta←conta+1
        fin_mientras
        Escribir("El total de la suma es:", suma)
Fin
```

Diagrama de flujo



Código en C++ del ejemplo anterior

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
        int conta=1,suma=0;
        clrscr();
        while(conta<=100)
        {
                suma=suma+conta;
        }
        cout<<"El total de la suma es:"<<suma;
        getch();

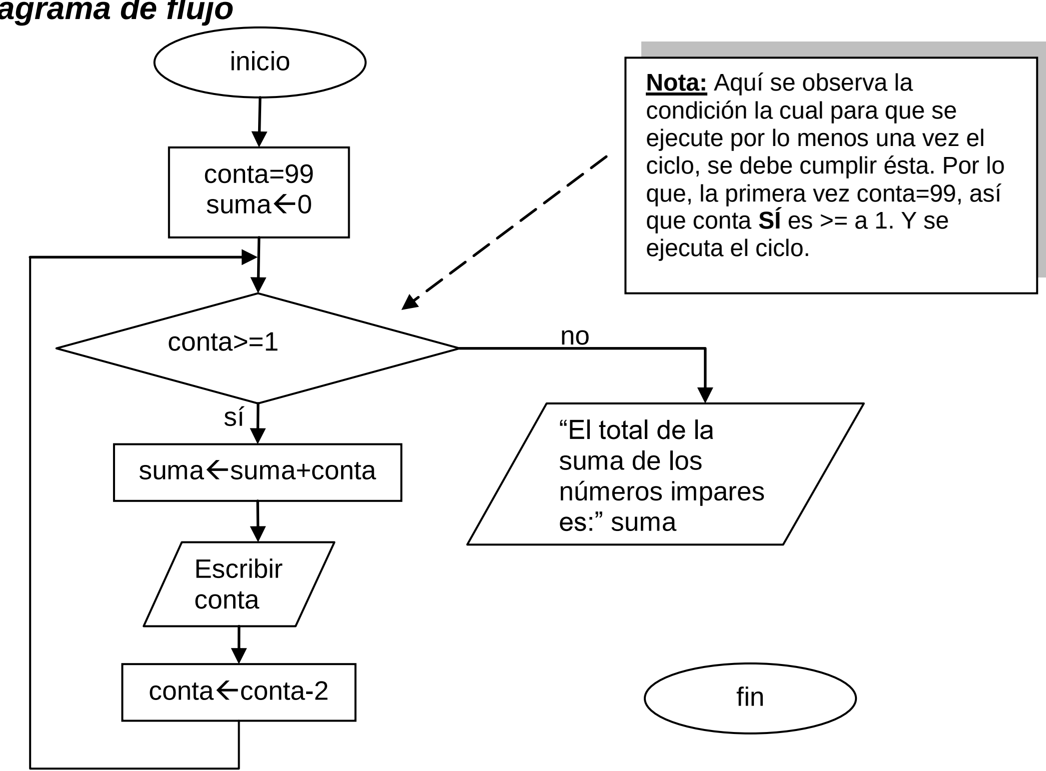
}
```

Ejemplo 2: Algoritmo que permita obtener la suma de todos los números impares del 100-1, así como escribir el correspondiente número primo comprendido en el rango citado.

Pseudocódigo

```
Inicio
        suma←0
        conta←99
        mientras conta>=1 hacer
                suma←suma+conta
                Escribir(conta)
                conta←conta-2
        fin_mientras
        Escribir("El total de la suma de los números impares es:", suma)
Fin
```

Diagrama de flujo



Código en C++ del ejemplo anterior

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
        int conta=99,suma=0;
        clrscr();
        while(conta>=1)
        {
                suma=suma+conta;
                cout<<"\n"<<conta;
                conta=conta-2;
        }
        cout<<" El total de la suma de los números impares es:"<<suma;
        getch();
}
```

Ejemplo 3: Algoritmo que permita calcular el promedio de calificaciones, con la característica de que el usuario podrá introducir tantas calificaciones como así desee, en el momento en que seleccione que no desea continuar capturando calificaciones, el algoritmo debe presentar el promedio de las calificaciones capturadas previamente.

Pseudocódigo

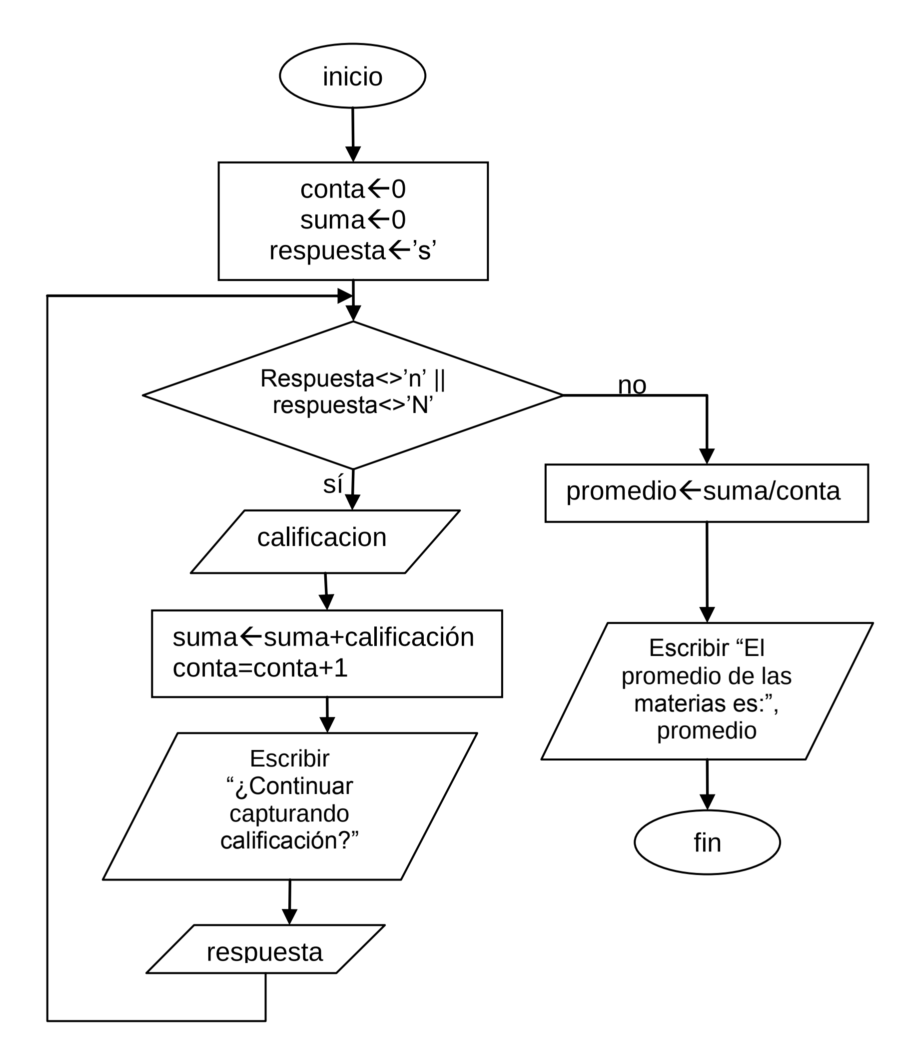
```
Inicio
        conta←0
        respuesta←'s'
        suma←0
        mientras respuesta<>'n' o respuesta<>'N' hacer
                leer(calificación)
                suma←suma+calificación
                conta←conta+1
                Escribir("¿Continuar capturando calificacion?")
                leer(respuesta)
        fin_mientras
```

promedio←suma/conta

escribir("El promedio de las materias es:",promedio)

fin

Diagrama de flujo



Código en C++ del ejemplo anterior

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
	float conta=0,suma=0,calificacion; char respuesta='s';
	clrscr();
	while(respuesta!='n' || respuesta!='N')
	{
		cout<<"\nCalificacion:";
		cin>>calificación;
		suma=suma+calificacion;
		conta=conta+1;
		cout<<"\n¿Continuar capturando calificación?";
		cin>>respuesta;
	}
	promedio=suma/conta;
	cout<<"\nEl promedio de las materias es:"<<promedio;
	getch();
}
```

3.5.2 DO…WHILE (HACER MIENTRAS)[30]

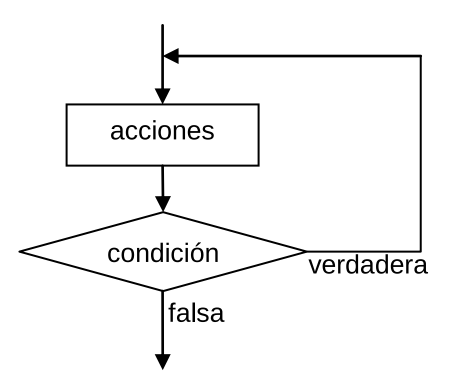
Joyanes Aguilar menciona que el bucle **hacer-mientras** es análogo al bucle **mientras** y el cuerpo del bucle se ejecuta una y otra vez mientras la condición (expresión *booleana*) es verdadera. Existe, sin embargo, una gran diferencia y es que el cuerpo del bucle está encerrado entre las palabras reservadas hacer y mientras, de modo que las sentencias de dicho cuerpo se ejecutan, al menos una vez, antes de que se evalúe la expresión *booleana*. En otras palabras, el cuerpo del bucle siempre se ejecuta, **al menos una vez**, incluso aunque la expresión *booleana* sea falsa.

---

[30] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 172.

Las representaciones gráficas son:

a) Diagrama de flujo



b) *Pseudocódigo*

hacer
<acciones>
mientras(<expresión>)

c) Sintaxis en C++:

```
do{
        instrucción(es);
}while(condición);
```

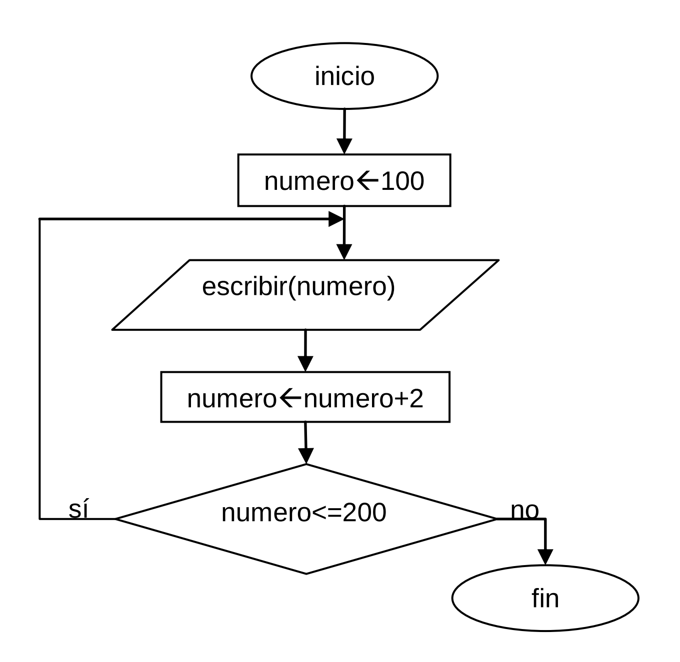
Ejemplo 1: Algoritmo que escriba de 2 en 2 los números comprendidos del 100 al 200.

Pseudocódigo

```
Inicio
        numero←100
        hacer
                escribir(numero)
                numero←numero+2
        mientras(numero<=200)
fin
```

Nota: Debido a que la condición se encuentra al final, esto garantiza que por lo menos una vez se ejecuta el conjunto de instrucciones de la estructura repetitiva.

Diagrama de flujo



Código en C++

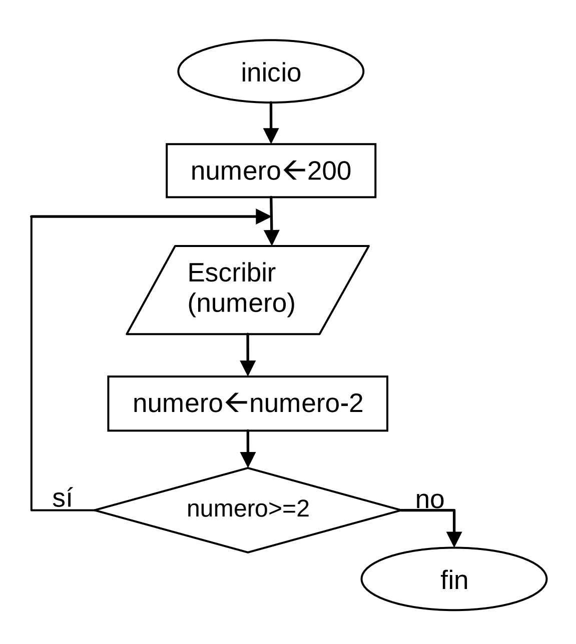
```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
	int numero=100;
	do{
		cout<<"\n"<<numero;
		numero=numero+2;
	}while(numero<=200);
	getch();
}
```

Ejemplo 2: Algoritmo que escribe los números 200-2 de 2 en 2.

Pseudocódigo

```
Inicio
	numero←200
	hacer
		escribir(numero)
		numero←numero-2
	mientras(numero>=2)
fin
```

Diagrama de flujo



Código en C++

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
	int numero=200;
	do{
		cout<<"\n"<<numero;
		numero=numero-2;
	}while(numero>=2);
	getch();
}
```

Ejemplo 3: Algoritmo que solicita un número y genere su correspondiente tabla de multiplicar desde el 1 hasta el 10. Y así sucesivamente hasta que el usuario ya no desee continuar generando tablas de multiplicar.

Pseudocódigo

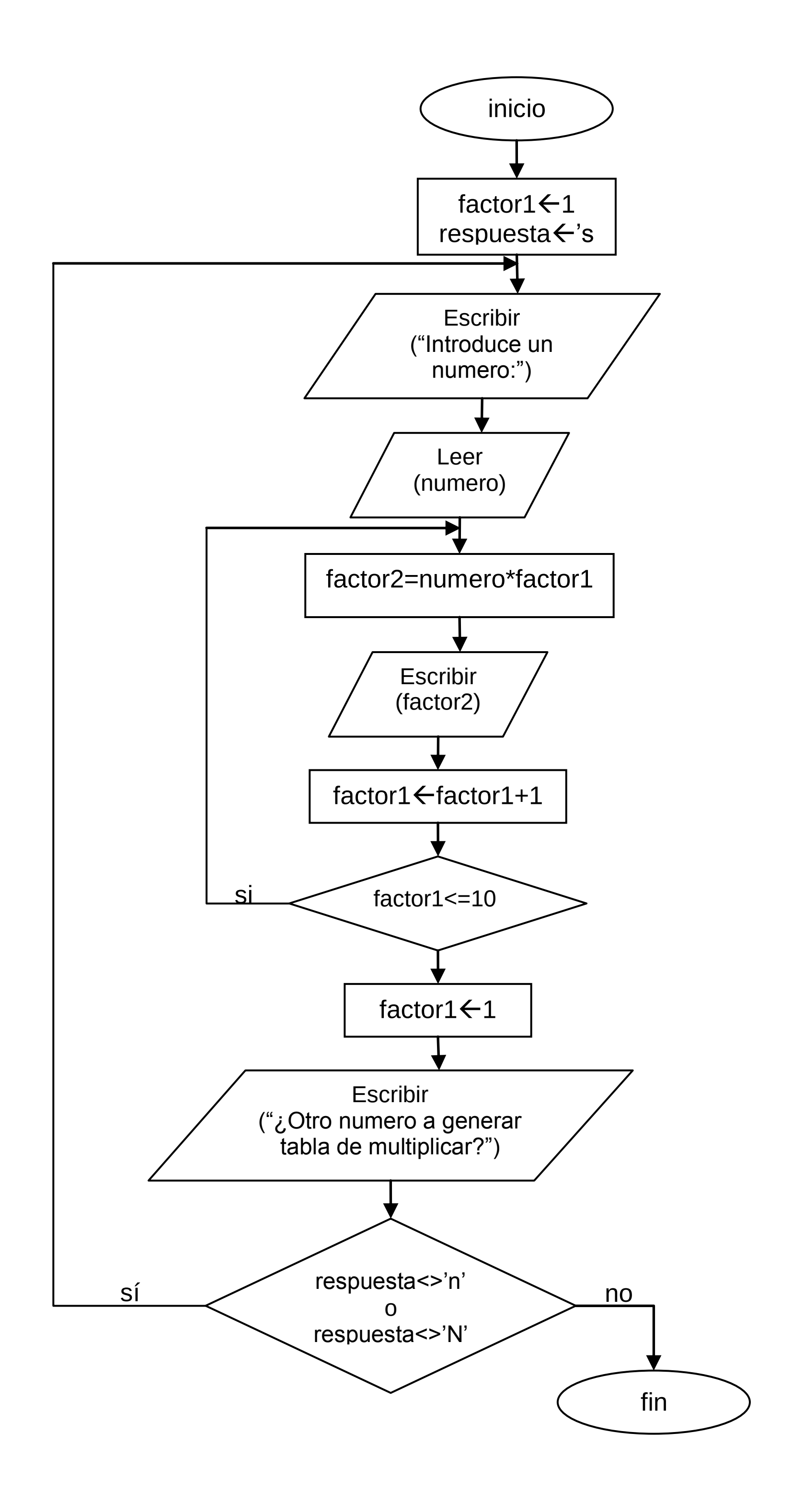
inicio

factor←1

respuesta←'s'

```
        hacer
                Escribir(“Introduce un número:”)
                Leer(numero)
                hacer
                        factor2=numero*factor1
                        escribir(factor2)
                        factor1=factor1+1
                mientras(factor1<=10)
                factor1←1
                Escribir(“¿Otro numero a generar tabla de multiplicar?”)
        mientras(respuesta<>’n’ o respuesta<>’N’)
fin
```

Diagrama de flujo


inicio
factor1←1
respuesta←'s
Escribir
("Introduce un
numero:")
Leer
(numero)
factor2=numero*factor1
Escribir
(factor2)
factor1←factor1+1
si
factor1<=10
factor1←1
Escribir
("¿Otro numero a generar
tabla de multiplicar?")
sí
respuesta<>'n'
o
respuesta<>'N'
no
fin

Código en C++

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
	int factor1=1,numero,factor2;
	char respuesta='s';
	do{
		cout<<"\n Introduce un numero:";cin>>numero;
		do{
			factor2=numero*factor1;
			cout<<"\n"<<factor2;
			factor1=factor1+1;
		}while(factor1<=10);
		factor1=1
		cout<<"\n¿Otro numero a generar tabla de multiplicar?";
		cin>>respuesta;
	}while(respuesta!='n' || respuesta!='N');
	getch();
}
```

3.5.3 ESTRUCTURA DESDE/PARA (FOR)[31]

En ocasiones se conoce de antemano el número de veces que se desean ejecutar las acciones de un *bucle*. En estos casos, en el que el número de iteraciones es fijo, se debe usar la estructura desde o para (for en inglés), señala *Joyanes Aguilar*. La estructura desde ejecuta las acciones del cuerpo del bucle un número especificado de veces y de modo automático controla el número de iteraciones o pasos a través del cuerpo del bucle.

[31] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 177.

La estructura **desde** comienza con un valor inicial de la variable índice y las acciones especificadas se ejecutan, a menos que el valor inicial sea mayor que el valor final. La variable índice se incrementa en uno y si este nuevo valor no excede al final, se ejecutan de nuevo las acciones. Por consiguiente, las acciones específicas en el bucle se ejecutan para cada valor de la variable índice desde el valor inicial hasta el valor final con el incremento de uno en uno.

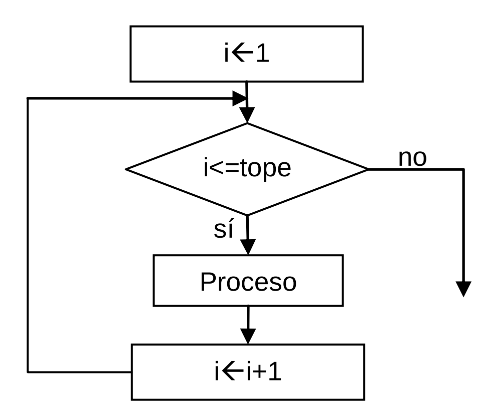
El incremento de la variable índice siempre es 1 si no se indica expresamente lo contrario. Dependiendo del tipo de lenguaje, es posible que el incremento sea distinto de uno, positivo, o negativo.

El formato de la estructura **desde** varía si se desea un incremento distinto a 1.

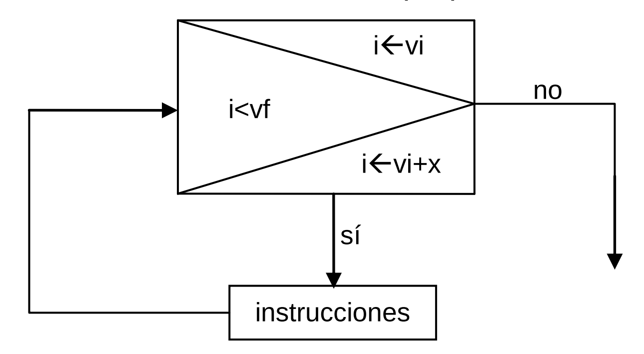
para v←vi **hasta** vf [incremento/decremento] hacer
    <acciones>
    .
    .
    .
Fin_para

Si el valor inicial de la variable índice, es menor que el valor final, los incrementos deben ser positivos, ya que en caso contrario la secuencia de las acciones no se ejecutaría. De igual modo, si el valor inicial es mayor que el valor final, no se efectuaría incremento por lo que se aplicaría un decremento.

Un bucle desde (for) se representa con los símbolos de proceso y decisión mediante un contador.



Es posible representar el bucle con símbolos propios.



Sintaxis en C++:

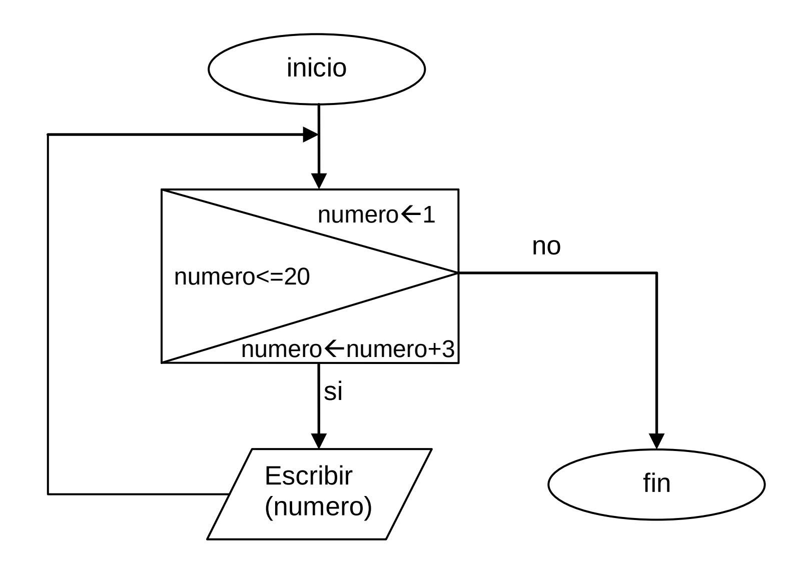
```
for(valor inicial; condición; incremento/decremento)
{
        Instrucción(es)
}
```

Ejemplo 1: Algoritmo que escribe en pantalla los números del 1-20 de 3 en 3.

Pseudocódigo Inicio

```
        para numero←1 hasta 20 incremento 3 hacer
                Escribir(numero)
        fin_para
fin
```

Diagrama de flujo



Código en C++

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
        int numero;
        clrscr();
        for(numero=0;numero<=20;numero=numero+3)
        {
                cout<<"\n"<<numero;
        }
        getch();
}
```

Ejemplo 2: Algoritmo que permita escribir los números del 200-1 de 1 en 1, así mismo al final obtener la suma correspondiente a los números generados.

Pseudocódigo

Inicio

    suma←0

    para numero←200 hasta 1 decremento 1 hacer

        suma←suma+numero

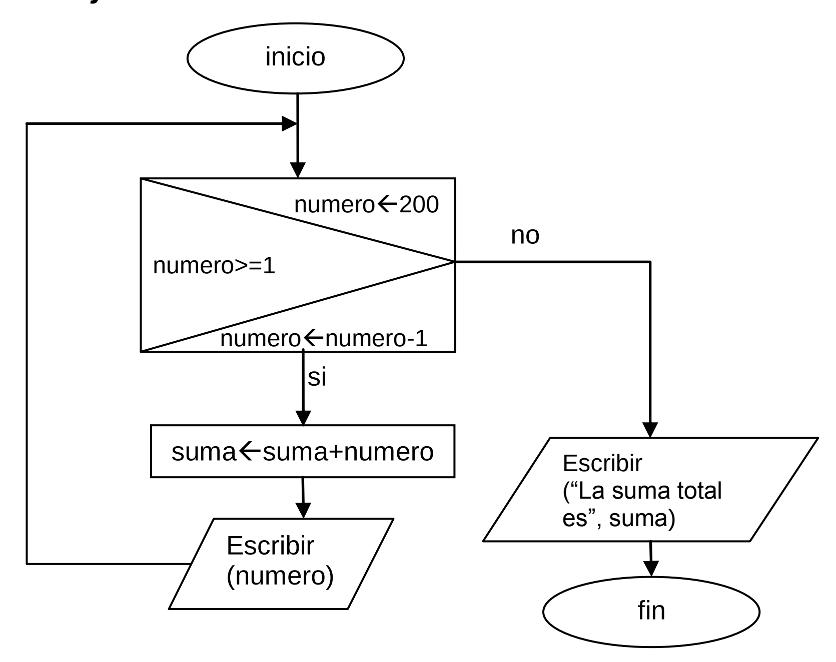
        Escribir(numero)

    fin_para

    Escribir(“La suma total es:”, suma)

fin

Diagrama de flujo



Código en C++

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
	int numero,suma;
	clrscr();
	for(numero=200;numero>=1;numero--)
	{
		suma=suma+numero;
		cout<<”\n”<<numero;
```

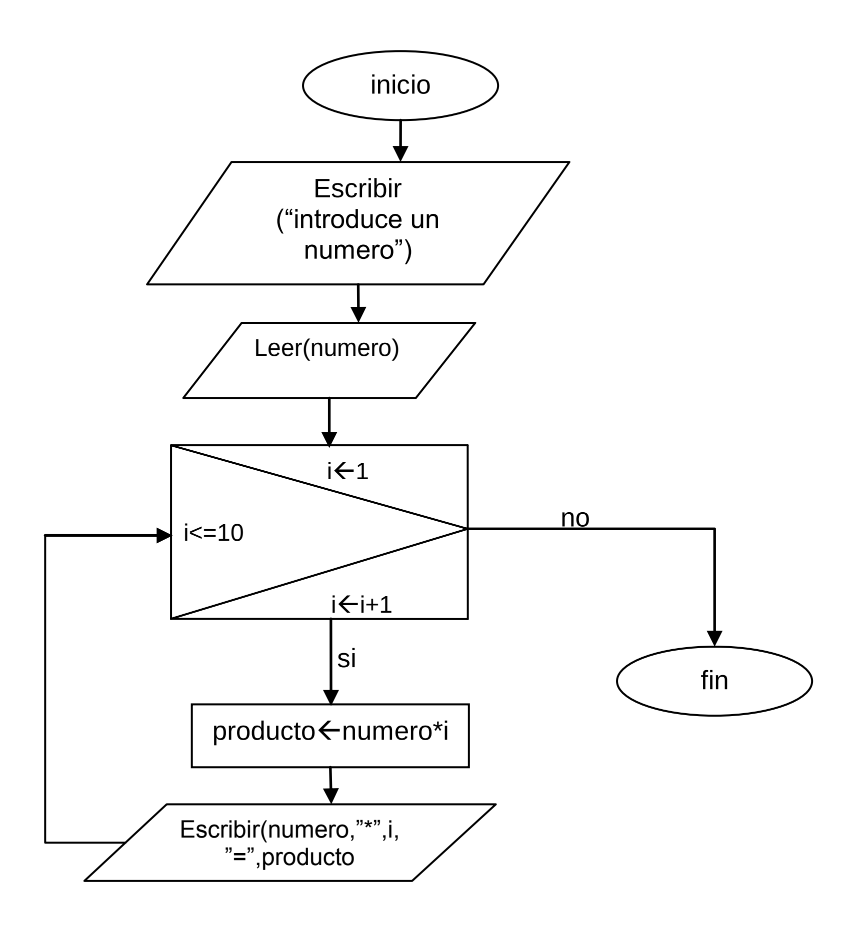
```
        }
        cout<<”\nLa suma total es:”<<suma;
        getch();
}
```

Ejemplo 3: Algoritmo que permita generar la tabla de multiplicar(1-10) de un número introducido previamente .

Pseudocódigo

```
Inicio
        Escribir(“introduce un numero”)
        Leer(numero)
        Para i←1 hasta 10 incremento 1 hacer
                producto←numero*1
                Escribir(numero,”*”,i,”=”,producto)
        fin_para
fin
```

Diagrama de flujo



Código en C++:

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
        int numero,i, producto;
        clrscr();
        cout<<"\nIntroduce un numero:";
        cin>>numero;
        for(i=1;i<=10;i++)
        {
                producto=numero*i;
                cout<<"\n"<<numero<<"*""<<i<<"="<<producto;
        }
        getch();
}
```

ACTIVIDADES DE APRENDIZAJE

1.- Realizar un cuadro sinóptico donde se describa los ciclos más importantes, así como las características de cada uno, señalar ventajas y desventajas. La entrega de la actividad es impresa. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

2.- Realizar los ejercicios contenidos en la lista de ejercicios de la sección 3.5.1

3.- Realizar los ejercicios contenidos en la lista de ejercicios de la sección 3.5.2

4.- Realizar los ejercicios contenidos en la lista de ejercicios de la sección 3.5.3

3.6. ARREGLOS.

Objetivo.

El alumno podrá definir el concepto de *arreglos* e identificará su aplicación en los algoritmos. Así mismo será capaz de resolver problemas básicos mediante diagramas de flujo.

Arreglos[32]

Joyanes Aguilar define a un Array, en inglés, (matriz, arreglo o vector) como un conjunto finito y ordenado de elementos homogéneos. La propiedad *"ordenado"* significa que el elemento primero, segundo, tercero,…, enésimo de un *array* puede ser identificado. Agrega que los elementos de un arreglo son homogéneos, es decir, del mismo tipo de datos. Un *array* puede estar compuesto de todos sus elementos de tipo cadena, otro puede tener todos sus elementos de tipo entero, etc. Los *arrays* se conocen como *matrices*, en matemáticas, y tablas en cálculos financieros.

Los arreglos según sus dimensiones se clasifican en:

- Unidimensionales
- De varias dimensiones
  - Arreglos bidimensionales (2 dimensiones, conocidos como tablas)
  - Arreglos multidimensionales (tridimensionales, por ejemplo.)

[32] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 249.

El tipo más simple de *arreglo* es el arreglo *unidimensional* o vector (matriz de una dimensión). Un vector de una dimensión denominado “***Numeros”*** que consta de n elementos se puede representar por la Figura 3.6.1.

Numeros(0)	Numeros(1)	…	Numeros(n)

Figura 3.6.1. Vector

El *subíndice o índice* de un elemento (1,2,…i,n) designa su posición en la ordenación del vector.

Es importante señalar que sólo el vector global tiene nombre (Numeros). Los elementos del vector se referencian por su *subíndice* o *índice*, es decir, su posición relativa en el valor.
El número de elementos de un vector se denomina *rango del vector.*

Cada elemento de un vector se puede procesar como si fuese una variable simple al ocupar una posición de memoria. Así
Numeros[15]←27
Almacena el valor entero o real 27 en la posición 15ª. del vector Numeros y la instrucción de salida
Escribir(Numeros(15))

Visualiza el valor almacenado en la posición 15ª. en este caso 27.

Esta propiedad significa que cada elemento de un vector, y posteriormente una tabla o matriz, es accesible directamente y es una de las *ventajas* más importantes de usar un vector: *almacenar un conjunto de datos.*

Operaciones con Vectores[33]

Las operaciones que se pueden realizar con vectores durante el proceso de resolución de un problema son:

- Asignación
- Lectura/escritura
- Recorrido(acceso secuencial)
- Actualizar(añadir, borrar, insertar)
- Ordenación,
- Búsqueda

Representación de un arreglo

```
Tipo
      Array[0..100] de entero:NUMERO
Var
      NUMERO:NU
```

Si se desea asignar los siguientes valores en un arreglo llamado DATOS, se efectuaría de la siguiente manera:

DATOS[0]←5

DATOS[1]←11

DATOS[2]←25

DATOS[3]←43

De la misma manera si se desea escribir los datos almacenados previamente en arreglo DATOS, se realizaría de la siguiente manera:

Escribir(DATOS[0])

Escribir(DATOS[1])

Escribir(DATOS[2])

Escribir(DATOS[3])

[33] [33] JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003. p. 252.

Como se logra apreciar este proceso es simple, pero se complica cuando existe la necesidad de almacenar una cantidad mayor de datos, por ejemplo, almacenar números en un arreglo de 100 elementos, las líneas de asignación se vuelven demasiadas, así como las líneas de escritura. Es por ello que para realizar las operaciones de asignación, lectura/escritura se recomienda apoyarse de las estructuras repetitivas, con el fin de que permitan manipular los subíndices correspondientes a las posiciones del arreglo en cuestión y esto permita el ahorro de procesos.

Por ejemplo, de un arreglo llamado NUMEROS de 100 elementos asignar el número 10 en cada uno de esos elementos:

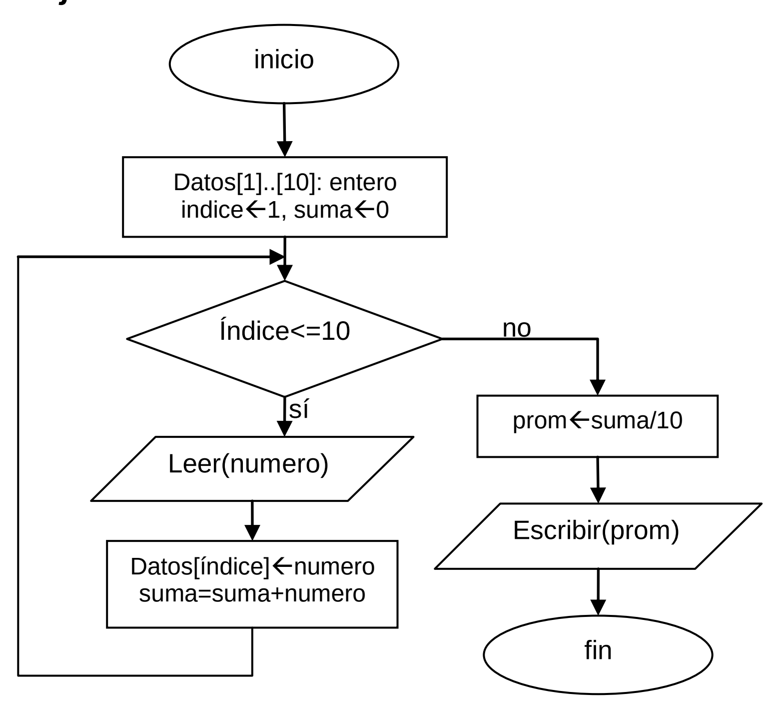
```
indice←0
Mientras índice<100 hacer
    NUMEROS[indice]←10
    indice←indice+1
Fin_mientras
```

Del arreglo anterior, con los elementos ya almacenados, leer el contenido de cada elemento y escribirlo.

```
indice←0
Mientras índice<100 hacer
    Escribir(NUMEROS)
    indice←indice+1
Fin_mientras
```

Ejemplo: Algoritmo que permita solicitar 10 elementos, los cuales serán almacenados en un arreglo, al final, debe visualizar el promedio de esos elementos.

Diagrama de flujo



Código en C++

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void main()
{
        int Datos[10],numero,suma=0,prom,índice
        while(índice<=10)
        {
                cout<<"\nIntroduce un numero:";
                cin>>numero;
                Datos[indice]=numero;
                suma=suma+numero;
        }
        prom=suma/10;
        cout<<"\nEl promedio es:"<<prom;
        clrscr();
}
```

ACTIVIDADES DE APRENDIZAJE

1.- Realizar un cuadro sinóptico de los tipos de arreglos, señalando sus características, y sintaxis. La entrega de la actividad es impresa. Mínimo dos cuartillas. Especificar bibliografía consultada. Considerar la limpieza y ortografía.

2.- Realizar un diagrama de flujo y programa que solicite diez números y los almacene en un arreglo y posteriormente imprima los datos. La entrega es impresa del diagrama de flujo, código y pantalla de salida una vez ejecutado el programa.

3.- Realizar un diagrama de flujo que solicite cinco números y los almacene en un arreglo llamado a, solicitar otro cinco números y almacenarlos en un arreglo llamado b. Posteriormente sumar el elemento 1 del arreglo a más el elemento 1 del arreglo b y el resultado almacenarlo en la posición 1 del arreglo c. Posteriormente sumar el elemento 2 del arreglo a más el elemento 2 del arreglo b y el resultado almacenarlo en la posición 2 del arreglo c, y así sucesivamente hasta sumar los 5 números. La entrega de la actividad es impresa. Considerar la limpieza y ortografía.

AUTOEVALUACIÓN

INSTRUCCIONES: Lee cuidadosamente y subraya la letra que corresponde a la palabra que complete la frase en cuestión.

1. Señala que las tareas suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el final del proceso.________________

a) Arreglos b) Contadores c) Ciclos d)Estructura secuencial

2. Se utilizan para tomar decisiones lógicas: esta es la razón que se suelan denominar también estructuras de decisión, condicionales o alternativas. _____________________________

a) Estructuras selectivas b) Estructura secuencial

c) Estructuras simples d) Estructuras complejas

3. Ejecuta una determinada acción cuando se cumple una determinada condición.__________________

a) For b) Do-while c)If-Then d) If-else

4. Es aquella variable que es utilizada en un ciclo repetitivo y tiene por objetivo almacenar valores cuyos incrementos o decrementos son en forma constante por cada iteración de ciclo o bucle en cuestión. ______________________________

a) Arreglo b) Contador c) Ciclo d) Bucle

5. Por lo general, se emplean en los ciclos para controlar el número de iteraciones en los mismos, o para almacenar totales de elementos. ______________

a) Arreglo b) Array c) Ciclo d) Contador

INSTRUCCIONES: Lee cuidadosamente a cada pregunta y subraya la respuesta que corresponda.

6. Es aquella variable que es utilizada en un ciclo repetitivo y tiene por objetivo almacenar valores cuyos incrementos o decrementos son en forma variable por cada iteración de ciclo o bucle en cuestión.

a) Arreglo b) Array c) Ciclo d) Acumulador

7. Es el hecho de repetir la ejecución de una secuencia de acciones.

a) Array b) Contador c) Iteración d) Bucle

8. Ejemplos de los ciclos repetitivos más comunes.

a) For, if-else, if-then –else b) While, do..while y for.

c) While, for, if-else d) Do…while, for, if-then-else

9. Puede estar compuesto de todos sus elementos de tipo cadena, otro puede tener todos sus elementos de tipo entero, etc.

a) Arreglo b) Contador c) Ciclo d) Array

10. ¿Cuál es una de las ventajas más importantes de usar un vector?

a) Almacenar tipos de datos b) Almacenar variables

c) Almacenar un conjunto de datos. d) Almacenar constantes

UNIDAD 4.

MODULARIDAD



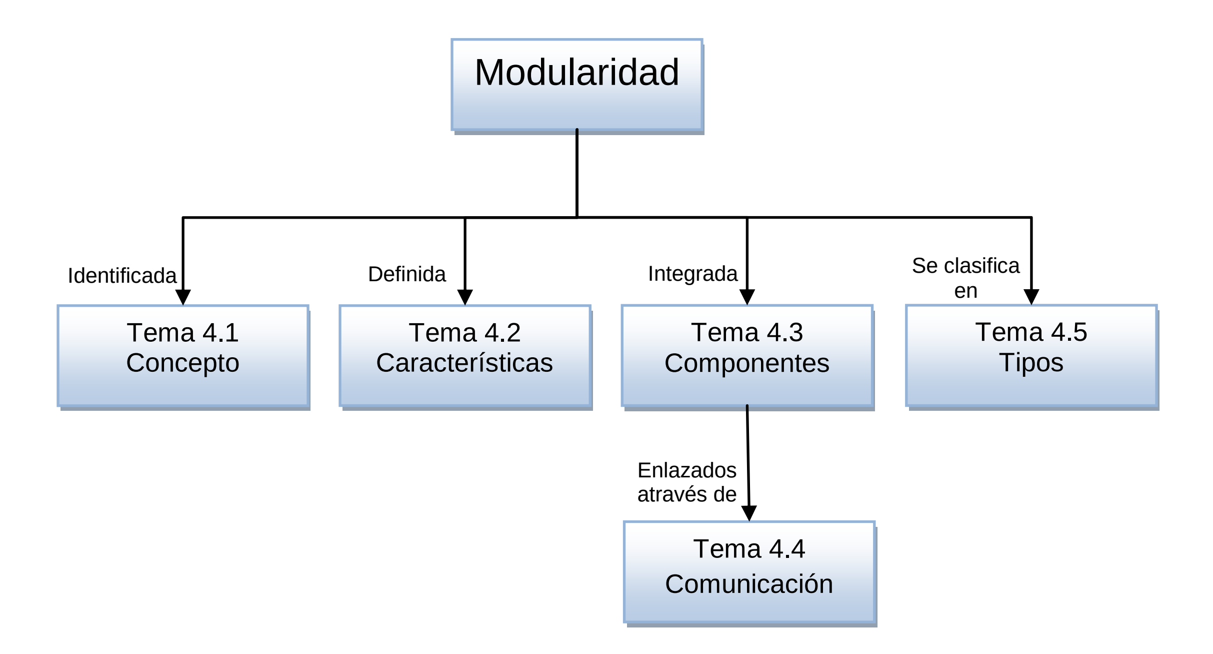
http://www.urbanity.es/foro/infraestructuras-inter/

OBJETIVO.

El alumno será capaz de evaluar la aplicación de la modularidad en problemas complejos para su mejor comprensión.

TEMARIO.

MAPA CONCEPTUAL



INTRODUCCIÓN

La ingeniería del software es la disciplina que se encarga de la creación y desarrollo de software; la principal tarea de la informática es aportar herramientas y procedimientos sobre los que se apoya esta disciplina en la construcción y desarrollo de proyectos.

La aplicación de métodos y técnicas para resolver los problemas, en la búsqueda de soluciones, es el origen de la programación modular que permite la descomposición de un problema en un conjunto de subproblemas independientes entre sí, más sencillos de resolver y que pueden ser tratados separadamente unos de otros.

Gracias a la modularidad se pueden probar los subprogramas o módulos de manera independiente, depurándose los errores antes de su inclusión en el programa principal y almacenándose para su posterior utilización cuantas veces se precise.

4.1. CONCEPTO

Objetivo

El participante podrá definir el concepto de modularidad e identificará sus aplicaciones en los algoritmos.

Behrouz, en su libro titulado *"Introducción a la ciencia de la computación"* afirma que la modularidad significa la división de un proyecto grande en partes más pequeñas que pueden entenderse y manejarse más fácilmente, es decir la división de un programa grande en partes más pequeñas que pueden comunicarse entre sí. [34]

En el mismo contexto, *Weitzenfeld* en su obra titulada "*Ingeniería del software orientada a objetos con UML, Java e Internet"* menciona que la modularidad permite dividir un sistema en componentes separados. Al contar con abstracciones de más alto nivel, la modularidad de un sistema se logra con base a componentes, de más alto nivel. Esto reduce el número final de componentes en un sistema y, a su vez, facilita su operación y mantenimiento.[35]

Badenas Carpio, ratifica que la modularidad posibilita la descomposición de un problema complejo en módulos más simples (análisis descendente). Además, proporciona la posibilidad de componer varios módulos ya escritos para otros fines permite construir nuevo software, de forma similar a como se utilizan funciones de bibliotecas en un lenguaje procedural.[36]

---

[34] A. FOROUZAN, Behrouz. Introducción a la ciencia de la computación, de la manipulación de datos a la teoría de la computación. International Thomson Editores. México, pág. 200.
[35] WEITZENFELD, Alfredo. Ingeniería del software orientada a objetos con UML, Java e Internet. Thomson, México 2005. Pág.25
[36] BADENAS CARPIO, Jorge, LLOPIO BORRÁS, José, COLTELL SIMÓN, Oscar. Curso práctico de programación en C y C++. Publicaciones de la Universidad Jaume I. DL., 1995, págs. 155-156.

Al estar compuesto de varios módulos, la compresión del sistema se facilita, ya que basta con comprender separadamente cada uno de los módulos. Un sistemas así concebido, está mejor armado contra las pequeñas modificaciones en las especificaciones, ya que estás no afectarán, normalmente, más que a un número reducido de módulos. Del mismo modo, el efecto de un error se limita también a pocos módulos. La ocultación de la información permite que la modificación en la implementación de un módulo que no afecte al interfaz, no provocarán ningún cambio en el resto del programa.[37]

La programación estructurada permite la escritura de programas que son fáciles de leer y modificar. Con este tipo de programación, el flujo lógico se gobierna por los tres tipos básicos de estructuras que son: las secuenciales, la de selección y las de repetición. Uno de los métodos más conocidos para resolver un problema es dividirlo en problemas más pequeños que se pueden llamar subproblemas. Esta técnica de subdivisión ha dado lugar a lo que se conoce como *programación modular*, la cual es uno de los métodos de diseño más flexibles y potentes que existen para mejorar la claridad de un programa en un conjunto de subproblemas más sencillos de resolver por separado. Esta técnica para programar la solución de problema se le suele llamar diseño descendente, metodología *“divide y vencerás”* o programación *arriba-abajo (top-down).* Un módulo, también llamado rutina o subrutina, es un procedimiento definido en un algoritmo que ejecuta una tarea específica. Un módulo puede ser llamado o invocado desde el módulo que describe el algoritmo principal (cualquier otro módulo) cuando su funcionalidad sea necesaria en un programa. Se puede entender que, con excepción del algoritmo principal, o módulo principal, un módulo depende siempre de otro.[38]

Sin embargo, los módulos que son invocados o llamados por un módulo principal deben ser independientes, en el sentido de que ningún módulo puede tener acceso directo a cualquier otro módulo. Esta idea de independencia

---

[37] Ídem.
[38] GÓMEZ DE SILVA GARZA, Andrés, ARIA BRISEÑO, Ignacio de Jesús. Introducción a la computación. Cengage Learning.1ª Edición 2008, págs.62-63

también se aplica a los propios submódulos de cualquier módulo. Además, de los resultados producidos por un módulo pueden ser utilizados por cualquier otro módulo a través del módulo que los llama, cuando se transfiere el control. El enfoque de diseño y programación modular es útil en dos casos:

1. Cuando existe un grupo de instrucciones o una tarea específica que debe ejecutarse en más de una ocasión.[39]
2. Cuando un problema es complejo o extenso y la solución se divide o segmenta en módulos que ejecutan partes o tareas específicas.

[39] Ídem.

ACTIVIDADES DE APRENDIZAJE

1.- Realizar una síntesis sobre el concepto de modularidad, consultando diversos autores. La entrega de la actividad es impresa. Especificar la bibliografía consultada. Considerar ortografía y limpieza.

4.2. CARACTERÍSTICAS

Objetivo

El estudiante será capaz de exponer las características, ventajas y desventajas de la modularidad.

La programación modular es una técnica de programación que permite[40]:

1.- Dividir la complejidad de un problema, convirtiendo problemas complejos en un conjunto de problemas más simples y por tanto más sencillos de implementar. Es la técnica del *"divide y vencerás".* Para llevar a cabo esta técnica se deberán identificar subproblemas o tareas más simples que la totalidad del problema a tratar. Al conjunto de acciones que implementan una determinada tarea de un problema se le denomina módulo. Los módulos deben cumplir dos características básicas para ser candidatos a una buena división del problema:

- Alta cohesión: las sentencias o instrucciones contenidas dentro de un módulo deben contribuir a la ejecución de la misma tarea.
- Bajo acoplamiento: la dependencia entre módulos debe ser los más pequeña posible. Ello implica que no debe haber datos compartidos entre los módulos, ni presunciones de cómo está implementado el otro módulo a la hora de ser ejecutado.

2.- Reutilizar el código: o conjunto de instrucciones de un programa en cualquier momento de la ejecución del mismo, e incluso crear "archivos de librería" que se podrán emplear en la implementación de diferentes programas.

[40] JORDÁ, Pedro Alonso, GARCÍA GRANADA, Fernando, ONAÍNDIA DE LA RIVAHERRERA, Eva. Diseño e implementación de programas en lenguaje C. Servicio de Publicaciones, España 1998. Págs. 157/158.

Evidentemente, la división de un problema en módulos no tiene por que ser ni única, ni obligatoria, pero si es claramente aconsejable a la hora de abordar un problema de cierta entidad.[41]

La programación modular, es llamada de diversas formas, en lenguaje estructurado C y C++ los módulos son conocidos como: *Funciones*, en **Basic** como *Subrutinas*, en **Pascal** son denominados *Procedimientos y funciones*, en **Fortran** *Subrutinas* y finalmente en **Cobol** son llamadas *Secciones*.

Joyanes en su obra titulada *"Fundamentos de programación, algoritmos, estructuras de datos y objetos"* afirma que en la etapa de análisis del proceso de programación se determina qué hace el programa. En la etapa de diseño se determina cómo hace el programa la tarea solicitada. Los métodos más eficaces para el proceso de diseño se basan en el conocimiento divide y vencerás. Es decir, la resolución de un programa complejo se realiza dividiendo el problema en subproblemas y a continuación a dividir estos subproblemas en otros de nivel más bajo, hasta que pueda ser implementada una solución en la computadora. Este método se conoce técnicamente como *diseño descendente (top-down) o modular*. Cada subprograma se resuelve mediante un módulo (subprograma) que tiene un solo punto de entrada y un solo punto de salida.[42]

Cualquier programa bien diseñado consta de un programa principal (el módulo de nivel más alto) que se llama subprogramas (módulos de nivel más bajo) que a su vez pueden llamar a otros subprogramas. Los programas estructurados de esta forma se dice que tienen un diseño modular y el método de romper el programa en módulos más pequeños se llama programación modular. Los módulos pueden ser planeados, codificados, comprobados y depurados independientemente (incluso por diferentes programadores) y a continuación combinarlos entre sí. El proceso implica la ejecución de los siguientes pasos hasta que el programa se termina:

---

[41] Ídem.
[42] JOYANES AGUILAR, Luis, Fundamentos de programación, algoritmos, estructuras de datos y objetos. Mc Graw Hill, 3ª Edición, España 2003. págs. 42

1. Programar un módulo.
2. Comprobar el módulo.
3. Si es necesario, depurar el módulo.
4. Combinar el módulo con los módulos anteriores.

El proceso que convierte, los resultados del análisis del problema en un diseño modular con refinamientos sucesivos que permiten una posterior traducción a un lenguaje se denomina diseño del algoritmo.[43]

Algunas de las ventajas de la programación modular, radica en que los módulos son independientes, el desarrollo de un programa se puede efectuar con mayor facilidad, ya que cada módulo se puede crear de forma aislada para que varios programadores trabajen simultáneamente en la confección de un algoritmo, repartiéndose las distintas partes del mismo. Así mismo, un módulo se puede modificar sin afectar a los demás. Gracias a la modularidad se pueden probar los módulos o subprogramas de manera independiente, depurándose sus errores antes de la inclusión en el programa principal y almacenándose para su posterior utilización cuantas veces se requiera.[44]

[43] Ibídem, pág. 43.
[44] [44] GÓMEZ DE SILVA GARZA, Andrés, ARIA BRISEÑO, Ignacio de Jesús. Introducción a la computación. Cengage Learning.1ª Edición 2008, Pág. 64

ACTIVIDADES DE APRENDIZAJE

1.- Realizar un cuadro sinóptico sobre las características de la modularidad. La entrega de la actividad es impresa. Especificar la bibliografía consultada. Considerar ortografía y limpieza.

4.3. COMPONENTES.

Objetivo

El alumno conocerá y podrá explicar los componentes de la modularidad.

Hablar de la programación modular, no es simplemente definir cada uno de sus componentes sino más bien interpretar y distinguir sus funciones y el uso apropiado, en este punto *Joyanes* confirma en su obra *"Fundamentos de programación, algoritmos, estructuras de datos y objetos"* que la programación modular es un de los métodos de diseño más flexibles y potentes para mejorar la productividad de un programa. En programación modular el programa se divide en módulos (partes independientes), cada una de las cuales ejecuta una única actividad o tarea y se codifican independientemente de otros módulos.[45] Cada programa contiene un módulo denominado programa principal, que controla todo lo que sucede; se transfiere el control a submódulos, de modo que ellos ejecutar sus funciones; sin embargo cada submódulo devuelve el control al módulo principal cuando se haya completado su tarea. Si la tarea asignada a cada submódulo es demasiado compleja; éste deberá romperse en otros módulos más pequeños. El proceso sucesivo de subdivisión de módulos continua hasta que cada modulo tenga solamente una tarea específica que ejecutar. Esta tarea puede ser entrada, salida, manipulación de datos, control de otros módulos o combinación de éstos. Un módulo puede transferir temporalmente (*bifurcar*) el control a otro módulo; sin embargo, cada módulo debe eventualmente devolver el control al módulo del cual se recibe originalmente el control.[46]

---

[45] [45] JOYANES AGUILAR, Luis, Fundamentos de programación, algoritmos, estructuras de datos y objetos. McGraw Hill, 3ª Edición, España 2003. págs. 49
[46] Ídem.

Los módulos son independientes en el sentido en que ningún módulo puede tener acceso directo a cualquier otro módulo excepto el módulo al que llama y sus propios submódulos. Sin embargo, los resultados producidos por un módulo pueden ser utilizados por cualquier otro módulo cuando se transfiera el control.

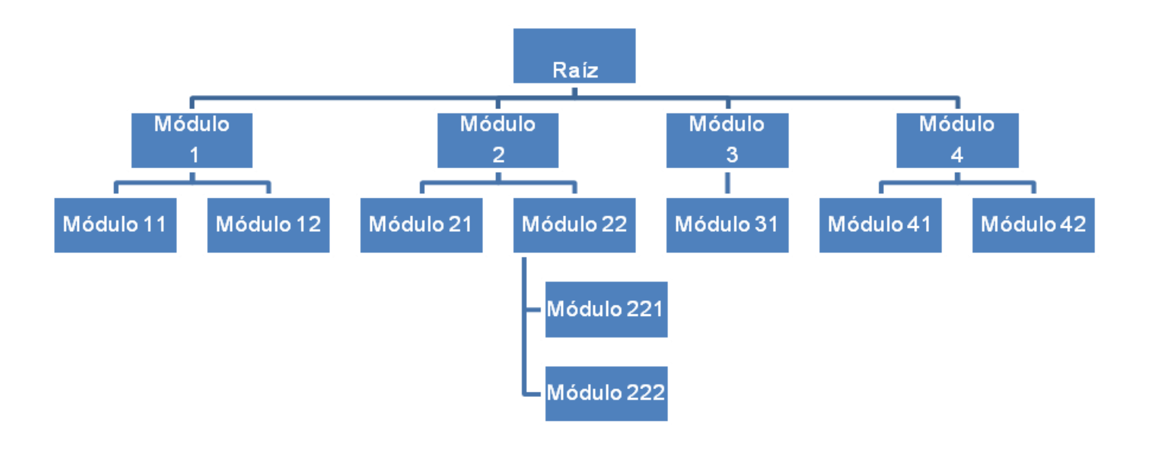


Figura 4.3. 1. Programación modular[47]

Dado que los módulos son independientes, diferentes programadores pueden trabajar simultáneamente en diferentes partes del mismo programa. Esto reducirá el tiempo del diseño del algoritmo y posterior codificación del programa. Además un módulo se puede codificar radicalmente sin afectar a otros módulos, incluso sin alterar su función principal. La descomposición de un programa en módulos independientes más simples se conoce también como el método de "divide y vencerás" (divide and conquer). Se diseña cada módulo con independencia de los demás, y siguiendo un método ascendente o descendente se llegará hasta la descomposición final del problema en módulos en forma jerárquica.

---

[47] Ídem.

ACTIVIDADES DE APRENDIZAJE

1.- Realizar un diagrama de flujo que imprima como opciones "1.- suma 2.-resta 3.-multiplicación 4.-division 5.- salir", al seccionar la opción correspondiente debe solicitar dos números, calcular la operación correspondiente e imprimir el resultado. Dar solución al problema empleando la modularidad. La actividad debe ser entregada impresa. Considerar limpieza y ortografía.

4.4. COMUNICACIÓN.

Objetivo.

El alumno reconocerá la importancia de la comunicación en la modularidad.

Cuando se construye un programa para resolver un problema, el problema completo se moldea, en primer lugar, como un único procedimiento. Este procedimiento de nivel superior se define entonces en términos de llamadas a otros procedimientos, que a su vez, se definen en términos de otros procedimientos creando una jerarquía de procedimientos. Este proceso continúa hasta que se alcanza una colección de procedimientos que ya no necesitan más refinamiento dado que los mismos se construyen totalmente en términos de sentencias en el lenguaje algorítmico. Esta es la razón de denominar a este método “refinamiento sucesivo” o “refinamiento descendiente top-down”. Entonces un programa completo se puede componer de un programa principal (main) y de otros procedimientos.[48]

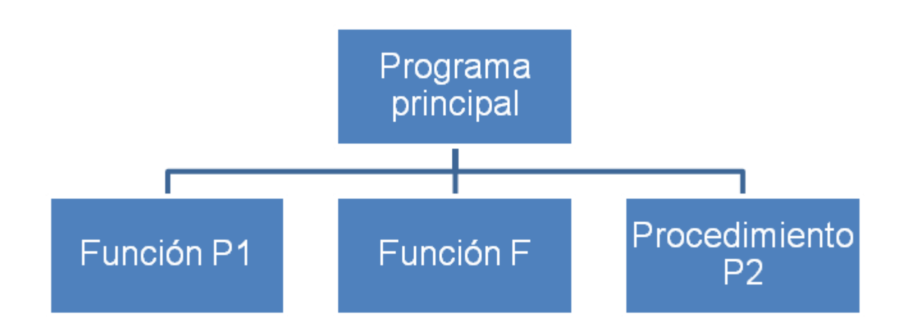


Figura 4.4.1. Un programa dividido en módulos independientes.[49]

[48] JOYANES AGUILAR, Luis, Fundamentos de programación, algoritmos, estructuras de datos y objetos. Mc Graw Hill, 3ª Edición, España 2003. Págs. 699.
[49] Ídem.

Cuando se escriben programas de tamaño y complejidad moderada, nos enfrentamos a la dificultad de escribir dichos programas. La solución para resolver estos problemas y, naturalmente, aquellos de mayor tamaño y complejidad, es recurrir a la modularidad mediante el diseño descendente. La filosofía del diseño descendente reside en que se descompone una tarea en sucesivos niveles de detalle. Para ello se divide el programa en módulos independientes, procedimientos, funciones y otros bloques de código. En la solución modular existe un módulo del más alto nivel que se va refinando en sentido descendente para encontrar módulos adicionales más pequeños. El resultado es una jerarquía de módulos; cada módulo se refina por los de bajo nivel que resuelve problemas más pequeños y contiene más detalles sobre los mismos. El proceso de refinamiento continúa hasta que los módulos de nivel inferior de la jerarquía sean tan simples como para introducirlos directamente a procedimientos, funciones y bloques de código en Pascal que resuelven problemas independientes muy pequeños. De hecho, cada módulo de nivel más bajo debe ejecutar una tarea bien definida. Estos módulos se denominan altamente cohesivos.

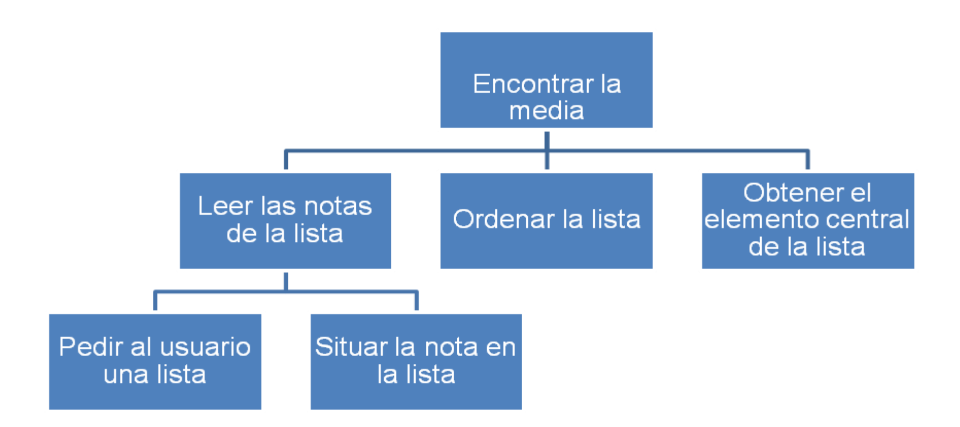


Figura 4.4.2. Diagrama bloques que muestra la jerarquía de módulos.[50]

Cada módulo se puede dividir en subtareas Por ejemplo, se puede refinar la tarea de leer las notas de una lista, dividiéndolo en dos subtareas. Por ejemplo,

---

[50] Ibídem, Pág. 700.

se puede refinar la tarea de leer las notas de la lista en otra dos subtareas: pedir al usuario una nota y situar una nota en la lista.

Modularidad mediante diseño descendente[51]

Un principio importante que ayuda a tratar la complejidad de un sistema es la modularidad. La descomposición del problema se realiza a través de un diseño descendente que a través de niveles sucesivos de refinamiento se obtendrán diferentes módulos. Normalmente los módulos de alto nivel especifican que acciones han de realizarse mientras que los módulos de bajo nivel definen cómo se realizan las acciones.

La programación modular tiene muchas ventajas. A medida que el tamaño de un programa crece, muchas tareas de programación se hacen más difíciles, la diferencia principal entre un programa modular pequeño y un programa modular grande es simplemente el número de módulos que cada uno contiene, ya que el trabajo con programas modulares es similar y sólo se ha de tener presente el modo en que unos módulos interactúan con otros. La modularidad tiene un impacto positivo en los siguientes aspectos de la programación:

- Construcción del programa. La descomposición de un programa en módulos permite que los diversos programadores trabajen de modo independiente en cada uno de sus módulos. El trabajo de módulos independientes convierte la tarea de escribir un programa grande en la tarea de escribir muchos programas pequeños.
- Depuración del programa. La depuración de programas grandes puede ser una tarea enorme, de modo que se facilitará esa tarea al centrarse en la depuración de pequeños programas más fáciles de verificar.
- Legibilidad. Los programas grandes son muy difíciles de leer, mientras que los programas modulares son muy fáciles de leer.

[51] Ibídem, Págs. 708-709

- Eliminación de código redundante. Otra ventaja del diseño modular es que se pueden identificar operaciones que suceden en muchas partes diferentes desprograma y se implementan como subprogramas. Esto significa que el código de una operación aparecerá sólo una vez, produciendo como resultado un aumento en la legibilidad y modificabilidad.

ACTIVIDADES DE APRENDIZAJE.

1.- Realizar una síntesis sobre la comunicación en la modularidad. Especificar bibliografía consultada. La entrega de la actividad es impresa. Considerar limpieza y ortografía.

4.5. TIPOS.

Objetivo

El estudiante identificará los tipos de modularidad.

Los módulos se clasifican en dos tipos principalmente: funciones y procedimientos.

Un procedimiento o una función consisten en un grupo de instrucciones, variables constantes, etc. Que están diseñados con un propósito particular y tiene su nombre propio. Se podrá decir que un procedimiento o una función son un “subprograma” del programa principal. La diferencia entre el procedimiento y función es la siguiente es un módulo de un programa que realiza una tarea específica pero que no devuelve ningún valor como resultado. En cambio, una función realiza también una tarea específica pero como resultado de ella se obtiene un valor que luego puede ser usado por el programa principal o por otro procedimiento o función. Un módulo encargado de borrar la pantalla sería un procedimiento, ya que no se devuelve ningún valor.

En cambio un módulo encargado de realizar una raíz cuadrada es una función, ya que después de realizar la raíz cuadrada aparece el resultado de ésta. Después de escribir un procedimiento o función, este puede ser ejecutado desde la función *main* o desde cualquier otro procedimiento o función. Para ello, se usa su propio nombre como una sola instrucción. A esto se le denomina llamada al procedimiento o función[52].

[52] DIDACT, SL. Manual de programación Lenguaje C++. 1ª Edición, Editorial MAD, S.L. España, 2005. Pág. 59.

PROCEDIMIENTOS[53]

Los procedimientos son subprogramas que pueden ser llamados desde la función main o desde otro procedimiento y que realizan una tarea determinada sin devolver ningún valor. Un ejemplo es:

```
Void Nombre procedimiento (parámetro1, parámetro2,…) {
Variables locales al procedimiento
….
Instrucciones del procedimiento.
}
```

La palabra *void* indica que el procedimiento no devuelve nada (si lo hiciera entonces sería una función). Los parámetros son variables que necesita el procedimiento para realizar su tarea. Si el procedimiento no necesitara parámetros, entonces entre paréntesis se colocaría también la palabra void.

Los procedimientos se pueden escribir antes o después de la función principal (main) del programa.

Un procedimiento o subrutina es un subprograma que ejecuta un proceso específico. Ningún valor está asociado con el nombre del procedimiento; por consiguiente, no puede ocurrir en una expresión. Un procedimiento se llama escribiendo su nombre, por ejemplo, SORT, para indicar que un procedimiento denominado SORT se va a usar. Cuando se invoca el procedimiento, los pasos que lo definen se ejecutan y a continuación se devuelve el control al programa que le llamó.

Los procedimientos y funciones son subprogramas cuyo diseño son similares; sin embargo, existen unas diferencias esenciales entre ellos:

1.- Un procedimiento es llamado desde el algoritmo o programa principal mediante su nombre y una lista de parámetros actuales, o bien con la

---

[53] JOYANES AGUILAR, Luis, Fundamentos de programación, algoritmos, estructuras de datos y objetos. Mc Graw Hill, 3ª Edición, España 2003. Págs. 207-227.

instrucción llamar_a (call). Al llamar al procedimiento se detiene momentáneamente el programa que se estuviera realizando y el control pasa al procedimiento llamado. Después que las acciones del procedimiento se ejecutan, se regresa a la acción inmediatamente siguiente a la que se llamó.

2.- Las funciones devuelven un valor, los procedimientos pueden devolver 0, 1 o n valores y en forma una lista de parámetros.

3.- El procedimiento se declara igual que la función, pero su nombre no está asociado a ninguno de los resultados que obtiene.

Los parámetros formales tienen –el mismo significado que en las funciones; los parámetros variables en aquellos lenguajes que los soportan, por ejemplo, Pascal están precedidos cada uno de ellos por la palabra var para designar que ellos obtendrán resultados del procedimiento en lugar de los valores actuales asociados a ellos. El procedimiento se llama mediante la instrucción:

[llamar_a] nombre [(lista de parámetros actuales)]

La palabra llamar_a (call) es opcional y su existencia depende del lenguaje de programación.

El ejemplo siguiente ilustra la definición y su de un procedimiento para realizar la división de dos números y obtener el cociente y el resto.

Varibles enteras: DIVIDENDO

DIVISOR

COCIENTE

Procedimiento

Procedimiento división (E entero: dividendo, divisor; S entero: cociente, resto)

Inicio

Cociente ← Dividendo DIV divisor

```
Resto ← Dividendo –cociente *divisor

Fin_procedimiento

Algoritmo principal

algoritmo aritmética

var

        entero: M,N,P,Q,S,T

inicio:

        leer (M,N)

        llamar_a division (m,N,P,Q)

        escribir (P,Q)

        llamar_a division (M*N-4, N+1,S,T)

        escribir (S,T)

fin
```

SUSTITUCIÓN DE ARGUMENTOS/PARÁMETROS[54]

La lista de parámetros, bien formales en el procedimiento o actuales (reales) en la llamada se conocen como lista de parámetros.

```
        Procedimiento demo
        .
        .
        .
        fin_procedimiento
```

[54] Ídem 53

O bien

Procedimiento_demo (lista de parámetros formales)

y la instrucción llamadora

llamar_a demo (lista de parámetros actuales)

Cuando se llama al procedimiento, cada parámetro formal toma como valor inicial el valor del correspondiente parámetro actual. En el ejemplo siguiente se indican la sustitución de parámetros y el orden correcto.

```
Algoritmo demo
          //definición del procedimiento
          entero: años
          real: numeros, tasa
inicio
…
          llamar_a calculo (numeros, años, tasa)
          …
Fin
Procedimiento calculo (S real: pl; E entero: p2; E real: p3)
inicio
          p3…p1…p2…p2
          fin_procedimiento
```

Las acciones sucesivas a realizar son las siguientes:

1. Los parámetros reales sustituyen a los parámetros formales
2. El cuerpo de la declaración del procedimiento se sustituye por la llamada del procedimiento.
3. Por último, se ejecutan las acciones escritas por el código resultante.

Las variables utilizadas en los programas principales y subprogramas se clasifican en dos tipos: variables *locales* y variables *globales.* Una variable local es aquella que está declarada y definida dentro de un subprograma, en el

sentido de que está dentro de ese subprograma y es distinta de las variables con el mismo nombre declaradas en cualquier parte del programa principal. El significado de una variable se confina al procedimiento en el que está declarada. Cuando otro subprograma utiliza el mismo nombre se refiere a una posición diferente en memoria. Se dice que tales variables son locales al subprograma en el que están declaradas. Una variable global es aquella que está declarada para el programa o algoritmo principal, del que dependen todos los subprogramas. La parte del programa/algoritmo en que una variable se define como ámbito (scope, en inglés).

COMUNICACIÓN CON SUBPROGRAMAS: PASO DE PARÁMETROS[55]

Cuando un programa llama a un subprograma, la información que comunica a través de la lista de parámetros y se establece una correspondencia automática entre los parámetros formales y actuales. Los parámetros actuales son "sustituidos" o "utilizados" en lugar de los parámetros formales. La declaración del subprograma se hace con:

```
Procedimiento nombre      (clase tipo_de_dato: F1)
                          (clase tipo_de_dato: F2)
                          ……………………….
                          (clase tipo_de_dato: Fn)
.
.
.
Fin_procedimiento
Y la llamada al subprograma con
Llamar_a nombre (A1, A2,……,An)
```

[55] Ídem 53

Donde F1,F2,….,Fn son los parámetros formales y A1, A2,…..,An los parámetros actuales. Existen dos métodos para establecer la correspondencia de parámetros:

1.- Correspondencia posicional. La correspondencia se establece aparejando los parámetros reales y formales según su posición en las listas: así Fi se corresponde con Ai, donde i=1, 2,….,n. Este método tiene algunas desventajas de legibilidad cuando el número de parámetros es grande.

2.- Correspondencia por el nombre explícito. También llamado método de paso de parámetros por nombre. En este método, en las llamadas se indica explícitamente la correspondencia entre los parámetros reales y formales. Este método se utiliza en Ada. Un ejemplo sería:

SUB (Y=>, X => 30); Que hace corresponder el parámetro actual B con el formal Y, y el parámetro actual 30 con el formal X durante la llamada de SUB.

PASO DE PARÁMETROS[56]

Los parámetros pueden ser clasificados:

Entradas: Proporcionan valores desde el programa que llama y que se utilizan dentro de un procedimiento. En los subprogramas función, las entradas son los argumentos en el sentido tradicional.

Salidas: Producen los resultados del subprograma; de nuevo si se utilizan el caso una función, este devuelve un valor calculado por dicha función, mientras que con procedimientos pueden calcularse cero, una o varias salidas.

---

[56] Ídem 53

Entradas/salidas: Un solo parámetro se utiliza para mandar argumentos a u programa y para devolver los resultados.

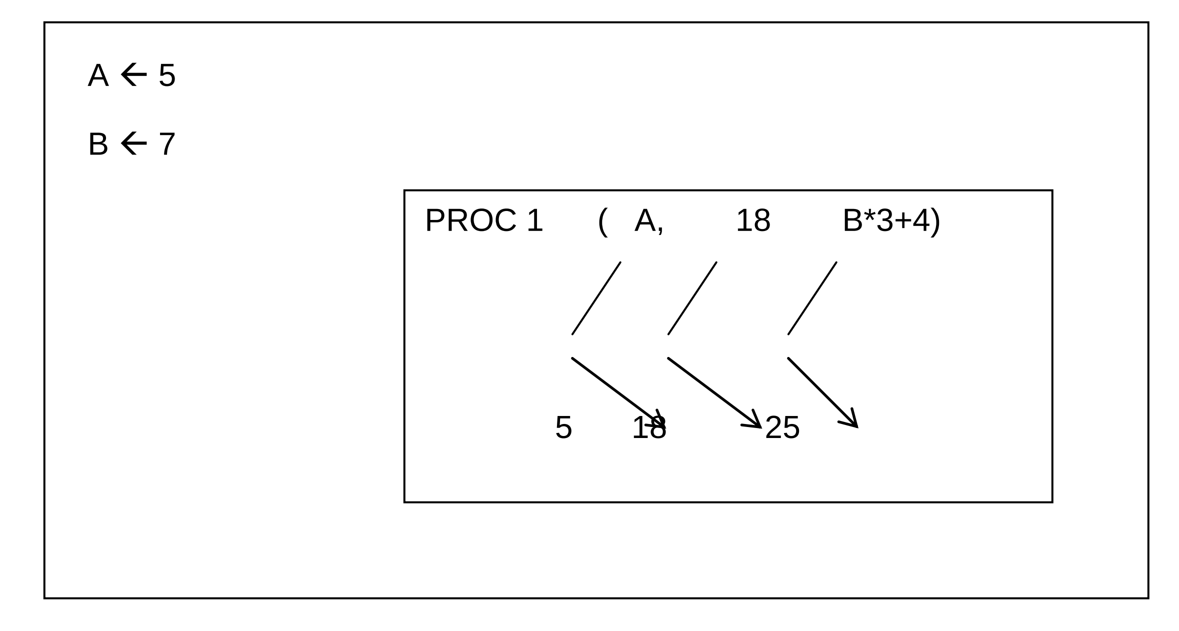
Los métodos más empleados para realizar el paso de parámetros son:

- Paso por valor.- También conocido por parámetro valor)
- Paso por referencia o dirección.- También conocido por parámetro variable)
- Paso por nombre
- Paso por resultado

PASO POR VALOR

El paso por valor se utiliza en muchos lenguajes de programación; por ejemplo, C, Modula-2, Pascal, Algol y Snobol. La razón de su popularidad es la analogía con los argumentos de una función, donde los valores se proporcionan en el orden de cálculo de resultados. Los parámetros se tratan como variables locales y los valores iniciales se proporcionan copiando los valores de los correspondientes argumentos.

Los parámetros formales- locales a función- reciben como valores iniciales los valores de los parámetros actuales y con ello se ejecutan las acciones descritas en el subprograma.



El mecanismo de paso se resume así:

Valor primer parámetro: A= 5

Valor segundo parámetro: constante =18

Valor tercer parámetro: expresión B*3+4=25

PASO POR REFERENCIA[57]

En numerosas ocasiones se requiere que ciertos parámetros sirvan como parámetros de salida, es decir, se devuelven los resultados a la unidad o programas que llama. Este método se denomina paso por referencia o también de llamada por dirección o variable. La unidad que llama pasa a la unidad llamada la dirección del parámetro actual (que está en el ámbito de la unidad llamante). Una referencia al correspondiente parámetro formal se trata como una referencia a la posición de memoria, cuya dirección se ha pasado. Entonces una variable pasada como un parámetro real es compartida, es decir, se puede modificar directamente por el subprograma.

La característica de este método se debe a su simplicidad y su analogía directa con la idea de que las variables tienen una posición de memoria asignada desde la cual se pueden obtener o actualizar sus valores. En este método los parámetros son de entrara/salida y los parámetros variables. Los parámetros valor y parámetros variables se suelen definir en la cabecera del subprograma. En el caso de lenguajes como Pascal, los parámetros variables deben ir precedidos por la palabra clave var.

programa muestra;

//parámetros actuales a y b, c y d paso por referencia

---

[57] Ídem 53

```
        Procedure prueba (var x,y:integer);
        Begin //procedimiento
// proceso de los valores de x e y
End;
Begin
.
.
.
    1. prueba (a,c);
    .
    .
    .
    2. prueba (b,d);
    .
    .
    .
    End.
```

La primera llamada en (1) produce que los parámetros a y c sean sustituidos y si los valores de x e y se modifican dentro de a o c en el algoritmo principal. De igual modo, b y d son sustituidos por x e y, cualquier modificación de x o y en el procedimiento afectará también al programa principal. La llamada por referencia es muy útil para programas donde se necesita la comunicación del valor en ambas direcciones.

FUNCIONES[58]

Matemáticamente una función es una operación que toma dos o más valores llamados argumentos y produce un valor denominado resultado. La declaración de una función requiere una serie de pasos que la definen. Una función como tal subalgoritmo o subprograma tiene una constitución similar a los algoritmos, por consiguiente constará de una cabecera que comenzará con el tipo del valor devuelto por la función, seguido de la palabra función y del nombre y argumentos de dicha función. A continuación irá el cuerpo de la función, que será una serie de acciones o instrucciones cuya ejecución hará que se asigne un valor al nombre de la función. Esto determina el valor particular del resultado que ha de devolverse al programa llamador.

```
<tipo_de_resultado> funcion <nombre_fun> (lista de parámetros)

[declaraciones locales]

Inicio
        <acciones>  //cuerpo de la función
        devolver (<expresión>)
fin_funcion
```

Lista de parámetros formales o argumentos, con uno o más argumentos de la siguiente forma:

({E | S | E/S} tipo_de_datoA: parámetro 1[,parámetro 2]…;

({E | S | E/S} tipo_de_datoA: parámetro x[,parámetro y]…;

Nombre_func.- Nombre asociado con la función, que será un nombre de identificador válido.

---

[58] Ídem 53

<acciones>.- Instrucciones que constituyen la definición de la función y que debe contener una única instrucción: devolver (<expresión>); expresión solo existe si la función se ha declarado con valor de retorno y expresión en el valor devuelto por la función.

Tipo_de_resultado.- Tipo del resultado que devuelve la función.

Sentencia devolver (return).- Se utiliza para regresar de una función (un método en programación orientada a objetos); devolver hace que el control del programa se transfiera al llamador de la función (método). Esta sentencia se puede utilizar para hacer que la ejecución regrese de nuevo al llamador de la función. La función devolver termina inmediatamente la función en la cual se ejecute.

INVOCACIÓN DE LAS FUNCIONES[59]

Una función puede ser llamada de la forma siguiente:

Nombre_función (lista de parámetros actuales)

Nombre_función //función que llama

Lista de parámetros actuales // constantes, variables, expresiones, valores de funciones, nombres de funciones o procedimientos.

Cada vez que se llama a una función desde el algoritmo principal se establece automáticamente una correspondencia entre los parámetros formales y los parámetros actuales. Debe haber exactamente el mismo número de parámetros actuales que de parámetros formales en la declaración de la función y se presupone una correspondencia uno a uno de izquierda a derecha entre los parámetros formales y los actuales. Una llamada a la función implica los siguientes pasos:

[59] Ídem 53

1.- A cada parámetro formal se le asignan el valor real de su correspondiente parámetro actual.

2. Se ejecuta el cuerpo de acciones de la función.

3.- Se devuelve el valor de la función y se retorna al punto de llamada.

Ejemplo: Realizar un programa haciendo uso de funciones, en donde se lleva a cabo una suma y resta. Diseñar así mismo el pseudocódigo y diagrama de flujo.

Pseudocódigo

1.- Inicio

2.- Lectura de datos (x,y)

3.- Seleccionar la opción

3.1.- Case1: suma

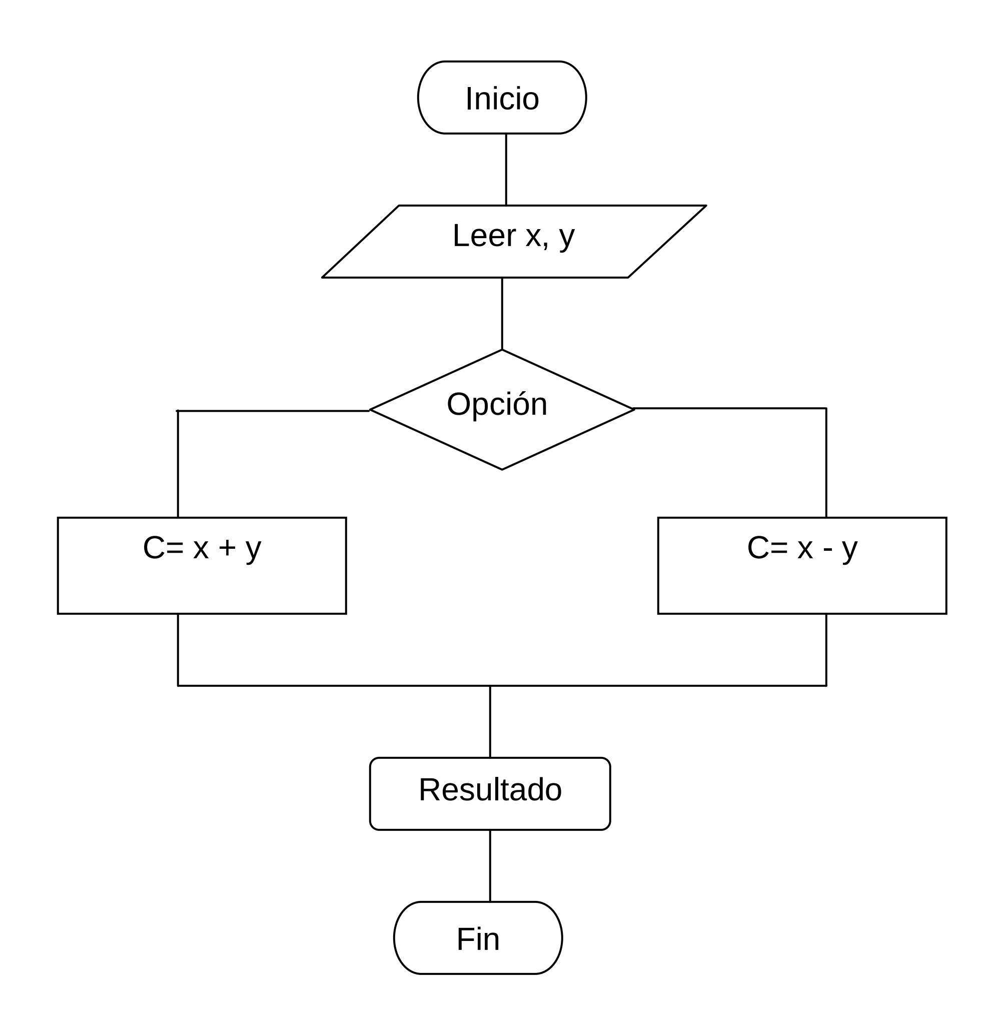
3.2.- Case2: resta

4.- Realizar las operaciones

5.- Visualizar el resultado (c)

6.- Fin

Diagrama de flujo:



Código en C++:

```
#include <stdio.h>
#include <conio.h>

int resta(int x,int y);
int suma(int x,int y);
int x,y,c;

void main()
{
```

```
int opcion;

        do
        {
        printf("Menu \n");
        printf("1 suma\n");
        printf("2 resta\n");
        printf("3 salir\n");
        scanf("%d",&opcion);

        switch(opcion)
        {
        case 1:suma();
        printf("Introduce el primer valor \n");
        scanf("%d",&x);
        printf("Introduce el segundo valor \n");
        scanf("%d",&y);
                break;
        case 2:resta();
        printf("Introduce el primer valor \n");
        scanf("%d",&x);
        printf("Introduce el segundo valor \n");
        scanf("%d",&y);
        break;
        }
} while(opcion!=3)

 suma( x, y)
{
        c=x+y;
   printf("%d",c);
```

```
}
 resta( x,  y)
{
    c=x-y;
printf("%d",c);
}

}
```

ACTIVIDADES DE APRENDIZAJE

1.- Realizar un cuadro sinóptico sobre los tipos de modularidad, señalar características, ventajas y desventajas. La entrega de la actividad es impresa. Mínimo dos cuartillas. Especificar bibliografía consultada. Considerar ortografía y limpieza.

2.- Realizar avance de proyecto. Dependiendo del tipo de proyecto, el catedrático solicita los contenidos apropiados para este avance. La entrega será impresa. Considerar ortografía y limpieza.

AUTOEVALUACIÓN

INSTRUCCIONES: Lee cuidadosamente y subraya la letra que corresponde a la palabra que complete la frase en cuestión.

1. A través del ____________ ___________ permite la descomposición de un problema complejo en módulos más simples.

a) Análisis descendente b) Análisis ascendente c) Lenguaje procedural

d) Ocultación modular

2. Al conjunto de acciones que implementan una determinada tarea de un problema se le denomina: ______________.

a) Top-down b) Módulo c) Función d) Procedimiento

3. La ___________ _____________ es uno de los métodos de diseño más flexibles y potentes que existen para mejorar la claridad de un programa en un conjunto de subproblemas más sencillos de resolver por separado.

a) Bajo acoplamiento b) Programación modular c) Alta cohesión

d) Baja cohesión

4. Los ____________ son subprogramas que pueden ser llamados desde la función main o desde otro procedimiento y que realizan una tarea determinada sin devolver ningún valor.

a) Funciones b) Módulos c) Procedimientos d) Void

5. Los módulos que son invocados o llamados por un módulo principal deben ser _________________, en el sentido de que ningún módulo puede tener acceso directo a cualquier otro módulo.

a) Independientes b) Dependientes c) Inversos d) Iguales

INSTRUCCIONES: Lee cuidadosamente a cada pregunta y subraya la respuesta que corresponda.

6. Los módulos se clasifican en dos tipos principalmente.

a) Análisis y diseño b) Rutinas y subrutinas

c) Funciones y procedimientos d) Divide y vencerás

7. Es llamado desde el algoritmo o programa principal mediante su nombre y una lista de parámetros actuales, o bien con la instrucción llamar_a (call).

a) Procedimiento b) Función c) VOID d) Módulo

8. Tipo de variable que está declarada para el programa o algoritmo principal, del que dependen todos los subprogramas.

a) Local b) Global c) SORT d) VOID

9. La característica de este método se debe a su simplicidad y su analogía directa con la idea de que las variables tienen una posición de memoria asignada desde la cual se pueden obtener o actualizar sus valores.

a) Paso por resultado b) Paso por nombre c) Paso por valor

d) Paso por referencia

10. Proporcionan valores desde el programa que llama y que se utilizan dentro de un procedimiento.

a) Parámetros de entrada b) Parámetros de salida c) Parámetro

d) Parámetros de Entrada/salida

PRÁCTICAS A DETALLE.

Práctica 1.

Realiza en el centro de cómputo, en donde el alumno identifique los elementos del entorno integrado de desarrollo, reglas de sintaxis del lenguaje C++. Entregar reporte de práctica, impreso mínimo tres cuartillas. En el cual se colocarán las pantallas correspondientes a los menús de C++, anotando las funciones que permiten cada una de las opciones de los menús. Anexar las reglas básicas de sintaxis (por ejemplo cuestiones a considerar cuando se declara una variable).

Práctica 2.

En esta práctica el alumno capturará el programa de ejemplo y anotará sus observaciones con respecto a los tipos de datos que puede introducir donde así corresponda. Anotar las ventajas y desventajas de los tipos de datos; así como señalar las operaciones aritméticas que se pueden llevar a cabo sobre los tipos de datos. El reporte se entregará impreso, mínimo dos cuartillas. Considerar ortografía y limpieza.

Práctica 3.

En asesoría del catedrático el alumno desarrollará el programa en lenguaje C++ de alguno de los algoritmos desarrollados por parte del alumno. El alumno de esta práctica entregará el algoritmo, el código del programa generado, así como la pantalla de salida cuando se ejecute el programa todo esto impreso.

LISTA DE EJERCICIOS.

Sección 3.5.1

1. Realizar el diagrama de flujo y programa que imprima del número 1 al 10 de 1 en 1. Empleando el ciclo while. La entrega de la actividad es impresa conteniendo el diagrama de flujo, código y pantalla de salida una vez ejecutado el programa. Considerar ortografía y limpieza.
2. Realizar un diagrama de flujo que imprima del número 20 al 2 de 1 en 1. Empleando el ciclo while. La entrega de la actividad es impresa. Considerar ortografía y limpieza.
3. Realizar un diagrama de flujo que imprima del número 1 al 100 de 2 en 2. Empleando el ciclo while. La entrega de la actividad es impresa. Considerar ortografía y limpieza.
4. Realizar un diagrama de flujo, empleando el ciclo while, que permita solicitar la calificación de 3 alumnos, los cuales tienen tres materias. Al final debe imprimir el promedio general de los alumnos. La entrega de la actividad será impresa. considerar ortografía y limpieza.
5. Realizar un diagrama de flujo que imprima del número 100 al 0 de 5 en 5. Empleando el ciclo while. La entrega de la actividad es impresa. Considerar ortografía y limpieza.
6. Realizar un diagrama de flujo que imprima del número 0 al 1000 de 3 en 3. Empleando el ciclo while. La entrega de la actividad es impresa. Considerar ortografía y limpieza.
7. Realizar un diagrama de flujo, empleando el ciclo while, que permita solicitar la calificación de 10 alumnos, los cuales tienen cinco materias. Al final debe imprimir el promedio general de los alumnos. Además de desplegar cuantos aprobaron y cuanto reprobaron. Teniendo en cuenta que la calificación mínima aprobatoria es 7.0 .La entrega de la actividad será impresa. Considerar ortografía y limpieza.

Sección 3.5.2

1. Realizar el diagrama de flujo y programa que imprima del número 1 al 10 de 1 en 1. empleando el ciclo do...while. La entrega de la actividad es impresa del diagrama de flujo, código y pantalla de salida una vez ejecutado el programa. Considerar ortografía y limpieza.
2. Realizar un diagrama de flujo que imprima del número 20 al 2 de 1 en 1.empleando el ciclo do...while. La entrega de la actividad es impresa. Considerar ortografía y limpieza.
3. Realizar un diagrama de flujo que imprima del número 1 al 100 de 2 en 2. Empleando el ciclo do...while. La entrega de la actividad es impresa. Considerar ortografía y limpieza.
4. Realizar un diagrama de flujo, empleando el ciclo do...while, que permita solicitar la calificación de 3 alumnos, los cuales tienen tres materias. al final debe imprimir el promedio general de los alumnos. La entrega de la actividad será impresa. Considerar ortografía y limpieza.
5. Realizar un diagrama de flujo, empleando el ciclo do..while, que represente el proceso de captura de productos en una nota de venta en un sistema. Se debe solicitar el número de productos a capturar, por cada producto solicitar el nombre, la cantidad a comprar del producto, su precio unitario; el proceso de solicitar datos se repite hasta que se llegue al total de productos especificados al principio. una vez capturados todos los productos imprimir el total a pagar, después de aplicar un descuento del 10%. la entrega de la actividad será impresa. considerar ortografía y limpieza.

1. Realizar un diagrama de flujo que imprima del número 200 al 2 de 4 en 4. Empleando el ciclo do...while. La entrega de la actividad es impresa. Considerar ortografía y limpieza.
2. Realizar un diagrama de flujo que imprima del número 1000 al 2000 de 2 en 2. Empleando el ciclo do...while. La entrega de la actividad es impresa. Considerar ortografía y limpieza.

3. Realizar un diagrama de flujo, empleando el ciclo do..while, que represente el proceso de captura de productos en una nota de venta en un sistema. Se debe solicitar el número de productos a capturar, por cada producto solicitar el nombre, la cantidad a comprar del producto, su precio unitario; el proceso de solicitar datos se repite hasta que se llegue al total de productos especificados al principio. Una vez capturados todos los productos imprimir el total a pagar, después de aplicar un descuento del 15%. a todos los totales mayores de 1000.00 . La entrega de la actividad será impresa. Considerar ortografía y limpieza.

Sección 3.5.3

1. Realizar un diagrama de flujo y programa que imprima del número 10 al 20 de 1 en 1. empleando el ciclo for. La entrega de la actividad es impresa del diagrama de flujo, código y pantalla de salida del programa una vez ejecutado. Considerar ortografía y limpieza.
2. Realizar un diagrama de flujo que imprima del número 30 al 2 de 1 en 1. Empleando el ciclo for. la entrega de la actividad es impresa. Considerar ortografía y limpieza.
3. Realizar un diagrama de flujo que imprima del número 2 al 200 de 2 en 2. empleando el ciclo for. La entrega de la actividad es impresa. Considerar ortografía y limpieza.
4. Realizar un diagrama de flujo y programa que imprima del número 500 al 200 de 1 en 1. Empleando el ciclo for. La entrega de la actividad es impresa del diagrama de flujo, código y pantalla de salida del programa una vez ejecutado. Considerar ortografía y limpieza.
5. Realizar un diagrama de flujo que imprima del número 300 al 2 de 1 en 1. Empleando el ciclo for. La entrega de la actividad es impresa. Considerar ortografía y limpieza.
6. Realizar un diagrama de flujo que imprima del número 30 al 200 de 2 en 2. Empleando el ciclo for. La entrega de la actividad es impresa. Considerar ortografía y limpieza.

BIBLIOGRAFÍA BÁSICA.

JOYANES AGUILAR, Luis. Fundamentos de programación. Mc Graw Hill. España. 2003.

JOYANES AGUILAR, Luis. Programación en C++. Algoritmos, Estructuras de datos y objetos. Mc Graw Hill. España. 2000. p.50.

NORTON, Peter. Introducción a la Computación. Mc Graw Hill. México. 3ra edición. 2000.

BEEKMAN, George. Computación & informática hoy. Una mirada a la tecnología del mañana. Addison-wesley. 1ra. Edición. 1995.

DEITEL, Deitel. Como programar en C++. Prentice-Hall. México. 1999.

GALVEZ, Javier. Algorítmica, diseño y análisis de algoritmos. Addison-Wesley. México.

LEVINE, Guillermo. Computación y Programación Moderna. Perspectiva integral de la informática. Addison-Wesley. México. 2001.

GLOSARIO.

Algoritmo: Conjunto de pasos ordenados o procedimientos para resolver un problema.

Analista de sistemas: Persona que analiza y diseña sistemas de software y provee mantenimiento y funciones de soporte para los usuarios.

Bit: (acrónimo de "BInary digiT", dígito binario) segmento individual de datos; ya sea un 0 o un 1.

Bloque: grupo contiguo de letras, palabras, oraciones, o párrafos seleccionados en un documento u hoja de cálculo con diversos propósitos, tales como mover, cortar o pegar.

C: lenguaje de programación desarrollado a principios de los años 70.

C++: lenguaje de programación extremadamente potente y eficiente desarrollado a principios de los años 80; superconjunto del lenguaje C con extensiones del lenguaje C con extensiones del lenguaje C con extensiones orientadas a objetos.

Carácter: 1) un número, letra, símbolo, o signo de puntuación; 2) una sola variable de ocho bits.

Char: abreviación de carácter, carácter.

Código máquina: código que reconoce la CPU como instrucciones en archivos de programas ejecutables.

Código fuente: instrucciones de programa creadas por los programadores cuando lo escriben.

Compilación: el primer paso en el proceso de conversión de archivos de código fuente de programa a programas ejecutables.

Compilador: programa que traduce un archivo de código fuente de programa a código objeto.

Entrada: datos sin procesar, suposiciones y fórmulas introducidas por computadora.

Flotante: variable que puede guardar números con puntos decimales.

Flujo de control: orden en que se ejecutan las declaraciones de un programa.

Información: 1) datos que han sido capturados y procesados por una computadora; 2) cualquier elemento no tangible que afecta a las empresas.

Instancia: se produce con la creación de un objeto perteneciente a una clase (instanciar una clase), que hereda entonces sus atributos, propiedades y métodos para ser usados dentro de un programa, ya sea como contenedores de datos o como partes funcionales del programa al contener en su interior funcionalidades de tratamiento de datos y procesamiento de la información que ha sido programada con anterioridad en la clase a la que pertenece.

Lenguaje de alto nivel: 1) originalmente, cualquier lenguaje más fácil de entender que el lenguaje de máquina; 2) actualmente usado para aquellos lenguajes más distantes del código máquina que el lenguaje ensamblador; usa palabras con mayor significado y frases, y provee facilidades para alterar el flujo de programas.

Memoria: uno de los dos componentes de procesamiento (junto con la CPU) de una computadora; área temporal de almacenamiento construido dentro del equipo de cómputo; lugar donde se guardan instrucciones y datos mientras son manipulados.

Objeto: unidad autocontenida definida dentro de una afirmación de un programa orientado a objetos; contiene tanto datos como funciones.

Salida: información generada por el procesamiento de datos de entrada.

Variable: parte de la memoria de una computadora que un programa reserva para su propio uso; número de bytes de memoria que puede contener un valor que puede cambiar.